Final Report

Title: Adaptive Anomaly Detection using Isolation Forest

AFOSR/AOARD Reference Number: FA2386-09-1-4014

AFOSR/AOARD Program Manager: **Hiroshi Motoda**, Ph.D.

Period of Performance: 2009

Submission Date: 30th December 2009

PI: Kai Ming Ting, Monash University, Gippsland Campus, Churchill, Victoria, Australia. Tel: +613 51226241

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
05 JAN 2010	**FInal**	**21-01-2009 to 20-01-2010**

4. TITLE AND SUBTITLE
Adaptive Anomaly Detection using Isolation Forest

5a. CONTRACT NUMBER
FA23860914014

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Kai Ming Ting

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Gippsland School of Information Technology, Monash University,Gippsland Campus, Churchill,Victoria 3842,Australia,au,3842

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AOARD, UNIT 45002, APO, AP, 96337-5002

10. SPONSOR/MONITOR'S ACRONYM(S)
AOARD

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AOARD-094014

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This project developed an adaptive anomaly detection system based on Isolation Forest, applicable to data stream which demands single-scan online algorithms with poly-logarithmic time and space complexities. The proposed system based on Half-Space Tree, an extension of Isolation Forest, is not only capable of detecting anomalies when the underlying concept changes gradually over time, but also capable of detecting abrupt changes in the underlying concepts. Half-Space Trees is significantly better than three existing state-of-the-art distance-based and density-based methods, in terms of detection accuracy, time complexity and memory requirement.

15. SUBJECT TERMS
Computer Science, Data Mining, Anomaly Detection

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**31**	

(2) Objective

This project aims to develop an adaptive anomaly detection system based on Isolation Forest, applicable to data stream which demands single-scan online algorithms with poly-logarithmic time and space complexities. The proposed system is not only capable of detecting anomalies when the underlying concept changes gradually over time, but also capable of detecting abrupt changes in the underlying concepts.

(3 & 4) Status of effort & Abstract

We have not only successfully achieved the objective of this project, but also uncovered a new ranking measure for anomaly detection. We have written two papers. The first paper reports the new ranking measure called mass and investigates its relationship to two commonly used ranking measures: distance and density; and it provides evidence that the proposed mass-based approach, called Half-Space Trees, is significantly better than three existing state-of-the-art distance-based and density-based methods, in terms of detection accuracy, time complexity and memory requirement. We also show that Isolation Forest is also a mass-based approach, uncovering the previously unknown principle underpinning the method we have reported in 2008. The second paper reports how the proposed mass-based approach can be adapted to deal with concept change, change detection and model adaptation in the context of data stream. There is no equivalent system which has all three capabilities that we are aware of in the literature. We have also identified three types of concept change, and revealed that only one type of change requires model update; whereas model update for other changes will degrade the detection accuracy. In addition, we have incorporated two change categories into the proposed method: transient change and permanent change, in order to detect abrupt changes in the underlying concepts.

(5) Personnel Supported

The grant is used to support a research assistant James Tan Swee Chuan, part-time for a period of 10 months.

(6) Publications

The following papers are submitted for publication as of 30th December 2009:

1. Ting, Tan & Liu. Mass: A new ranking measure for anomaly detection. Submitted to IEEE Transactions on Knowledge and Data Engineering.
2. Tan, Ting & Liu. Half-Space Trees: Amortised *O(1)* anomaly detection algorithm in evolving data stream. To be submitted to the Sixteenth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington D.C. July 25-28 2010 (submission due 2 Feb 2010.)

(10) Archival Documentation

Attached are the two papers listed in item (6) above.

Other items are not applicable.

Gippsland School of Information Technology

Technical Report

Title: Mass: A New Ranking Measure for Anomaly Detection

Authors: Kai Ming Ting, Swee Chuan Tan, Fei Tony Liu

TR No.: TR2009/1

Department: Gippsland School of Information Technology

Faculty: Information Technology

Mass: A New Ranking Measure for Anomaly Detection

Kai Ming Ting, James Tan Swee Chuan and Fei Tony Liu
Gippsland School of Information Technology,
Monash University, Australia.
kaiming.ting@infotech.monash.edu.au

Abstract—Ranking measure is of prime importance in anomaly detection tasks because it is required to rank the instances from the most anomalous to the most normal. This paper investigates the underlying assumptions and definitions used for ranking in existing anomaly detection methods; and it has three aims: First, we show evidence that the two commonly used ranking measures—distance and density—cannot accurately rank clustered anomalies in anomaly detection tasks. We introduce a new measure—mass, which can accurately rank both scattered and clustered anomalies. Second, we propose a definition of anomaly based on this new measure and contrast it with the current definitions based on distance and density. We identify the strengths and weaknesses of these definitions, and demonstrate the advantages of the new definition based on mass. Third, we propose a mass-based approach for anomaly detection called Half-Space Tree and show that it performs favourably to three existing state-of-the-art distance-based and density-based anomaly detection methods in term of detection accuracy, runtime and memory space requirements.

Index Terms—Anomaly Detection, Ranking Measures, Mass, Scattered Anomalies, Clustered Anomalies.

1 Introduction

Anomalies are data patterns that have different data characteristics from normal instances. The detection of anomalies often provides critical actionable information and brings about significant impact on the task at hand. For example, anomalies in credit card transactions could signify a fraudulent use of credit cards [2]. Abnormal patient condition could indicate a disease outbreak in a specific area [34]. An unusual computer network traffic pattern could signify an unauthorised access [15]. These applications demand anomaly detectors with high detection rates and fast execution.

Anomaly detection usually involves ranking a set of instances, from the most anomalous instances at the top of the ranked list to the most normal instances at the bottom. Since this is a ranking problem, the ranking measure is of prime importance to the success of the method. Distance and density are the two most commonly used basic ranking measures, and we refer to anomaly detection methods based on them as distance-based and density-based methods, respectively, in this paper. There are many variants of these two basic measures such as Local Outlier Factor [9]—a density measure based on k-nearest neighbours; and multi-granularity deviation factor [26]—a density measure based on regions with a fixed radius.

This paper investigates the underlying assumptions and definitions for ranking anomalies which underpin many existing anomaly detection approaches. We show that these assumptions and definitions are only good for detecting *scattered anomalies* but fail to detect *clustered anomalies*—a problem known as 'masking effect' [25] in statistics literature [7, 27]. These anomalies group in cluster(s) and are difficult to detect because of their close promixity to each other, thus have high density—the exact opposite to the assumptions that anomalies are far from normal instances and have low density in order for distance-based and density-based methods to function well.

It is important to incorporate the ability to detect clustered anomalies in detectors. The inability to detect clustered anomalies by existing methods are a 'loophole' that can be exploited by fraudsters or organsisms that have evolved to evade detection. For example, credit card fraudsters might incur a short burst of abnormal transactions that masquerate their abnormality with their high occurence. This has high financial cost if they are not detected quickly. Similarly, viruses might mutate to exploit this weakness that can cause many lives if they evade detection for a prolong period of time.

This paper has three aims. First, we propose a new basic ranking measure called **mass** for anomaly detection and demonstrate that it has different properties from the two commonly used ranking measures: distance and density. Second, we propose a definition of anomaly based on this new measure and contrast it with the current definitions based on distance and density. We identify the strengths and weaknesses of these definitions, and demonstrate the advantages of the new definition based on mass. Third, we devise a mass-based approach for anomaly detection, and we show evidence that mass is indeed a better ranking measure than distance and density[1]; and the mass-based approach has a better performance in terms of detection accuracy, runtime and memory space requirements than existing state-of-the-art distance-based anomaly detectors such as ORCA [8] and one-class SVM [31], and density-based anomaly detector LOF [9].

The rest of the paper is organised as follows. Section 2 introduces the proposed new ranking measure: mass. Section

1. Note that a dichotomy into anomalies and normal points is a special case of ranking, which is used by some anomaly detectors.

3 describes various definitions used for ranking anomalies and contrasts the difference with the definition based on mass. Section 4 introduces the mass-based approach to anomaly detection. Section 5 provides an empirical evaluation, and Section 6 stipulates a comparison of time and space complexities among anomaly detection methods. Section 7 describes the connection of the proposed mass-based approach to a closely related method. We describe the related work and conclusions in the last two sections.

2 A NEW RANKING MEASURE: MASS

Distance and density are the two basic ranking measures commonly used to rank instances for anomaly detection. Distance is usually defined based on some distance or similarity metric; the typical distance metrics are Euclidean, L_p-norm or Mahalanobis distance measures.

The common definition for density is the number of data point per unit space[2]. The other possible definition for density is based on a kernel function i.e., kernel density estimation in which a kernel function is placed in each point and the overall density is the sum of the kernel function applied to all points. In other words, density is defined against some standard (unit space or kernel function).

Definition 1 : **Data mass** or **mass**, the new ranking measure we proposed, is defined as the number of points in a region; and two groups of data can have the same mass regardless of the characteristics of the regions (e.g., density, shape and size).

Let X a feature space and R_i a subspace, $R_i \subset X$. The mass in subspace R_i is the total number of instances in that subspace, denoted as $m(R_i)$; The basis function for mass distribution is a rectangular function which is defined as

$$f(x) = \begin{cases} m(R_i) & \text{if } x \in R_i \\ 0 & \text{otherwise} \end{cases}$$

A mass distribution based on $f(x)$ has the following properties:

- The height of the distribution indicates the mass of the data in their local region rather than denseness.
- Any two uni-modal density distributions which possess the same data mass will have the same height in their mass distributions, regardless of their densities.

All the above-mentioned properties[3] are illustrated using a simple example in Figure 1, in contrast to density distribution estimated using histogram.

A 'smoothed' mass distribution can be obtained by using an ensemble method from the rectangular basis function $f(x)$, without explicitly defining a region.

Let $\{R_1^x, R_2^x, \ldots, R_t^x\}$ the set of t subspaces which all cover point x, i.e., $\bigcap_i R_i^x \supseteq \{x\}$; and each subspace R_i^x is a random 'variant' of each other.

2. A variant is defined as number of points per unit distance when k-nearest neighbours are used to measure density—a ratio of k and the sum of distances to each of the k nearest neighbours.

3. It is interesting to note that there is only one way to compute mass, given a region; but there are many ways to compute density or distance, depending on the metric used.

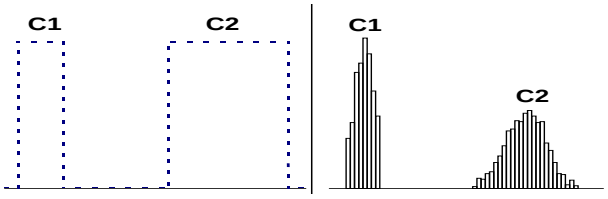


Fig. 1. Mass versus density: Two regions having the same number of instances (1000 data points each), with different densities (sd=0.125 and 1.0, respectively), have the same mass shown as rectangular functions of the same height. The histograms depict the differing densities of the two regions.

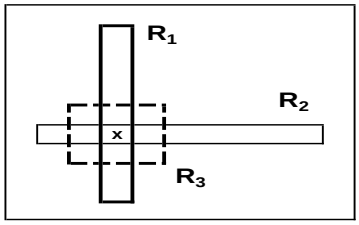


Fig. 2. An example of three subspaces covering x.

Definition 2 : The **mass distribution**, $mass(x)$, estimates mass at point x by a simple averaging over all masses in all subspaces R_i^x covering x as follows.

$$mass(x) = \frac{\sum_{i=1}^{t} m(R_i^x)}{t} \quad (1)$$

An example of R_i^x is shown in Figure 2. Note that this mass definition applies to any data distribution because no assumption of data distribution in R_i^x is made. In practice, R_i^x can be generated randomly without explicitly searching for one. We will show such a mass-based approach in Section 4.

Note that the resultant mass distribution derived from $mass(x)$ is a 'smoothed' distribution, which stipulates a gradation of the mass between adjacent points. Examples of the 'smoothed' distribution will be showed in the next section (in Figure 3).

$mass(x)$ can then be used to rank instances according to their masses—**instances with low mass are more likely to be anomalies than instances with high mass**. In this paper, we show that the ranking provided by mass is good for detecting both scattered anomalies and clustered anomalies; but the ranking provided by either density or distance is only good for scattered anomalies. These will be discussed in Section 3.2. We will first discuss how ranking measures are used for anomaly detection in the next section.

3 RANKING FOR ANOMALY DETECTION

3.1 Definitions of anomalies

There are a number of accepted definitions for anomalies. Some prevalent definitions are:

'An observation which deviates so much from other observations as to arouse suspicion that they were generated by a different mechanism' [18].

'An observation which appears to be inconsistent with the remainder of that set of data' [7].

The key property of anomalies highlighted in these and other definitions is: '**different**', and the implicit assumption is that anomalies are '**few**' with respect to the normal points. We will do our analysis in terms of 'few' and 'different' in the following.

We focus on definitions used in the distance-based and density-based anomaly detection approaches because they are the prevalent approaches. The definitions commonly used are:

(i) **D-Distance Anomalies** are data points which have fewer than p neighboring points within a distance D [22].

(ii) **kth NN Distance Anomalies** are the top-ranked instances whose distance to the kth nearest neighbor is greatest [29].

(iii) **Average kNN Distance Anomalies** are the top-ranked instances whose average distance to the k nearest neighbors is greatest [4].

(iv) **Density-based Anomalies** are instances which are in regions of low density or low local/relative density.

Definition (i) uses distance D to differentiate one region from another and uses the number of instances within each region to define 'few'. This is very restrictive because the regions are constrained to a specific shape defined by the distance metric employed, and D is fixed globally. Definitions (ii) and (iii) measure 'different' in terms of distance, and k controls the number of instances to be considered as 'few'—an (anomaly) region with instances more than k will not be considered as anomalies using this definition. Definition (iv) correctly refers to 'few' when low density regions have small number of instances; but this definition assigns low density regions which have many instances as anomaly regions, and high density regions which have few instances as normal regions—both of which are counter-intuitive. These definitions are good for detecting scattered anomalies (regions with one or two instances thus low density which are far from normal instances) but fail to detect clustered anomalies (regions with few instances but high density.)

In contrast, the definition based on data mass is

(v) '**Mass-based Anomalies** are instances which are in regions of low mass, regardless of density, shape and size of the regions.'

'Low mass' refers specifically to 'few', and 'mass' by its definition refers to a region that is 'different' from other regions.

We argue that the definition of mass-based anomalies can better capsulate the true nature of anomalies than either distance-based or density-based definitions.

3.2 Scattered and clustered anomalies

Here we examine the ability of the five definitions mentioned in the last section to rank scattered anomalies as well as clustered anomalies. We first contrast the density-based and mass-based definitions; and then show that the distance-based definition has the same 'deficiency' as the density-based definition.

Let region $R(m, d)$ has m instances and density d, calculated by a density estimation method. Assume that we have a set of t regions $R_1, R_2, R_3, ..., R_t$. Consider the following four scenarios:

(a) The regions have the same d but increasing masses.

(b) The regions range from dense small regions to sparse large regions: $R_1(m, td), R_2(2m, (t-1)d), R_3(3m, (t-2)d), ..., R_t(tm, d)$.

(c) The regions range from sparse small regions to dense large regions: $R_1(m, d), R_2(2m, (2d), R_3(3m, 3d), ..., R_n(tm, td)$.

(d) The regions have the same m but increasing densities.

The series of masses and densities for each scenario is shown in Table 1.

Scenario	Mass	Density
(a)	$m, 2m, 3m, ..., tm$	$d, d, d, ..., d$
(b)	$m, 2m, 3m, ..., tm$	$td, (t-1)d, (t-2)d, ..., d$
(c)	$m, 2m, 3m, ..., tm$	$d, 2d, 3d, ..., td$
(d)	$m, m, m, ..., m$	$d, 2d, 3d, ..., td$

TABLE 1
Series of masses and densities of t regions in four different scenarios: (a), (b), (c), (d).

We highlight the following observations:

- The definition for density-based anomalies ranks all regions equally in scenario (a), and ranks the sparse large regions in front of the dense small regions in scenario (b). Both of the above rankings are counter-intuitive—small regions (because of being 'few') are obviously the more likely candidates of anomalies than large regions. Scenario (b) is the case in which there are clustered anomalies which might be denser than the normal regions.
- The definition for mass-based anomalies ranks small regions before the large regions in both scenarios (a) and (b), regardless of the densities.
- Scenario (c) is an easy case for all measures for which the rankings are identical and in the right order. When $m = 1$, this scenario is the one which has scattered anomalies and dense normal regions.
- In scenario (d), it is arguable that the regions should be ranked at all in terms of anomaly ranking: for large m, they are all unlikely to be anomalies; for small m, say 1 or 2 instances, the density estimation is inaccurate, rendering the ranking meaningless.

Figure 3 shows an example in which the anomaly definitions based on distance and density (defined using k nearest neighbours) will fail to rank the dense small region (having 20 points denoted as C1) ahead of the sparse large region (having 1000 points denoted as C2); whereas the mass measure ranks them correctly. The 'smoothed' mass distribution is obtained from an ensemble of 50 mass-based models called HS*-Trees. This mass-based ensemble approach is to be introduced in the next section.

4 Mass-based approach

This section has three aims. First, we propose a practical anomaly detector which provides an effective ranking using mass. Second, we compare the three ranking measures using

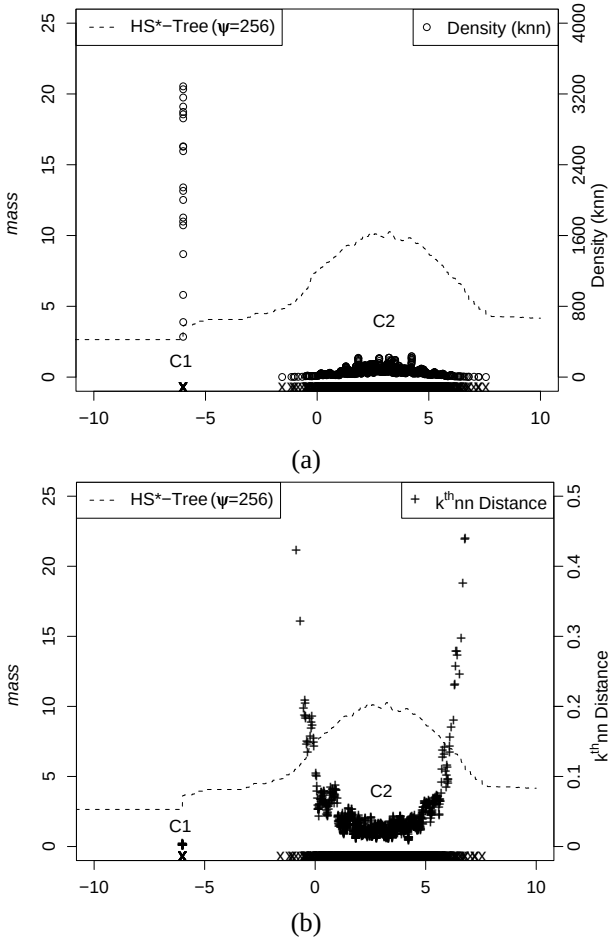
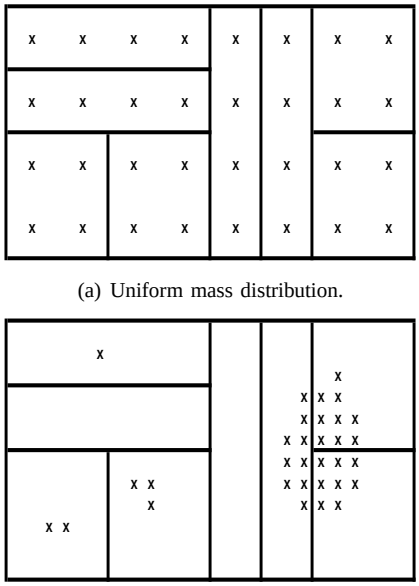


Fig. 3. A comparison of different ranking measure distributions provided by mass, (a) density (kNN) and (b) k^{th}NN distance using an example of a dense small region C1 of 20 points (on the left) and a sparse large region C2 of 1000 points (on the right). Each 'X' symbol in the last row denotes one point in one-dimension (x-axis); note that C1 appears to have only one 'X' because the 20 points are very close to each other, i.e., very dense.

the same anomaly detector and show that mass is effective for both scattered and clustered anomalies whereas the other two measures are not. Third, we provide empirical evidence that the proposed mass-based approach performs better than the state-of-the-art anomaly detectors which employ density and distance ranking measures.

4.1 Half-Space Tree

The motivation of the proposed method, Half-Space Tree, comes from the fact that equal-size subspaces contain the same mass in a space with uniform mass distribution, regardless of the shapes of the subspaces. This is shown in Figure 4(a), where the space enveloped by the data is split into equal-size half-spaces recursively three times into eight subdivisions. Note that the shapes of the eight subdivisions may be different because the splits at the same level may not use the same attribute.



(a) Uniform mass distribution.

(b) Non-uniform mass distribution.

Fig. 4. Half-space subdivisions of: (a) uniform mass distribution; and (b) non-uniform mass distribution.

The binary half-space split ensures that every split produces two equal-size half-spaces, each containing exactly half of the mass before the split under a uniform mass distribution. This characteristic enables us to compute the relationship between any subdivisions easily. For example, the mass in every subdivision shown in Figure 4(a) is the same, and it is equivalent to the original mass divided by 2^3 because three levels of binary half-space subdivisions have been applied. A deviation from the uniform mass distribution allows us to rank the subdivisions based on mass. Figure 4(b) provides such an example in which a ranking of subdivisions based on mass provides an order of the degrees of anomaly in each subdivision.

In the following two subsections, we first provide definitions for the proposed Half-Space Tree method and its two different variants; then, we present the Half-Space Tree algorithm.

4.2 Definitions

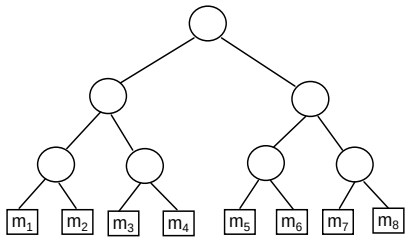
Definition 3 : Half-Space Tree is a binary tree in which each internal node makes a half-space split into two equal-space subdivisions, and each external node terminates further splits. All nodes record the mass of the training data in their own subdivisions.

Let $R[i]$ be a half-space subdivision at depth level i with mass $m(R[i])$ or short for $m[i]$.

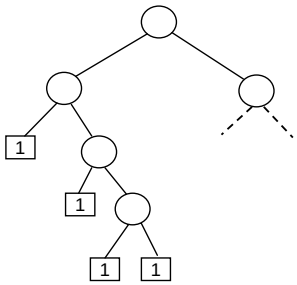
Definition 4 : Equivalence of mass between any two subdivisions is expressed with reference to $m[i=0]$ at depth level=0 (the root) of a Half-Space Tree.

Under uniform mass distribution, the mass at level i is related to mass at level 0 as follows:

$$m[i=0] = m[i] \times 2^i,$$



(a) HS-Tree.



(b) HS*-Tree.

Fig. 5. Half-Space Tree: (a) HS-Tree: An HS-Tree for the data shown in Figure 4a has $m_i = 4, \forall i$, which are $m[\ell = 3]$ (i.e., mass at level 3). (b) HS*-Tree: This is an example of a special case of HS*-Tree when the size limit is set to 1.

or masses between any subdivisons at levels i and j are related as follows:

$$m[i] \times 2^i = m[j] \times 2^j.$$

Under non-uniform mass distribution, the following inequality establishes an ordering between subdivisions at different levels:

$$m[i] \times 2^i < m[j] \times 2^j.$$

We employ the above property to rank instances and define the anomaly score s based on (augmented) mass for Half-Space Tree as follows.

$$s(x) = m[\ell] \times 2^\ell, \tag{2}$$

where ℓ is the depth level of an external node with $m[\ell]$ instances in which a test instance x falls into.

The anomaly score is based on mass $m[\ell]$ only if a Half-Space tree has all external nodes at the same depth level. The score is based on augmented mass, $m[\ell] \times 2^\ell$, if the external nodes have differing depth levels. We describe two such variants of Half-Space Tree below.

HS-Tree: Ranking based on mass only. The first variant, HS-Tree, builds a balanced binary tree structure which makes a half-space split at each internal node and all external nodes have the same depth. The number of training instances falls into each external node is recorded and it is used for scoring in testing. An example of HS-Tree is shown in Figure 5(a).

HS*-Tree: Ranking based on augmented mass. Unlike HS-Tree, the second variant, HS*-Tree, has differing depth levels. The anomaly score for HS*-Tree is Equation (2) in order to account for different depths. We call this score *augmented mass*, as the mass is augmented in the calculation by the level of subdivision in HS*-Tree, as opposed to mass only in HS-Tree.

In a special case of HS*-Tree, an external node only terminates when the training data size is 1. Here the anomaly score reduces to depth level only, i.e., 2^ℓ or simply ℓ. In other words, **the depth level becomes a proxy to mass in HS*-Tree** when the size limit is set to 1. An example of HS*-Tree, when the size limit is set to 1, is shown in Figure 5(b).

Since the two variants have similar performance, we will focus on HS*-Tree only in the rest of this paper; and use the terms 'mass' and 'augmented mass' interchangably, unless a distinction is required.

Ensemble. The proposed method uses a random subsample to build one Half-Space Tree, and multiple Half-Space Trees are constructed from different random subsamples (without replacement) to form an ensemble.

4.3 Algorithm for HS*-Trees

The proposed method estimates a mass distribution efficiently, even in a multi-dimensional space, without density or distance calculations or clustering. The method employs an ensemble to produce a 'smoothed' mass distribution which is an average of augmented mass from all trees in the ensemble, where every mass estimate for an instance x from a tree is assuming a rectangular function over the region in which x resides. This method requires only a small data sample to produce a mass distribution that is suitable for anomaly detection.

Training. The first step in generating an HS*-Tree from a data sample is to establish a working space. An internal node in HS*-Tree is created by randomly selecting a dimension, and a half-space split on this dimension is established for this node. Then, the process is repeated for each branch until a size limit (or a depth limit) is reached to form an external node. The training instances at the external node at depth level ℓ form the mass $m[\ell]$ to be used during the testing process. The training procedures for an ensemble of HS*-Trees are shown in Algorithms 1 and 2.

Algorithm 1 : HS*-Trees(X, t, ψ, S, h)

Inputs: X - input data, t - number of trees, ψ - sub-sampling size, S - data size limit at external node, h - maximum depth limit

Output: F - a set of t HS*-Trees

1: $SizeLimit \leftarrow S$
2: $MaxDepthLimit \leftarrow h$
3: **Initialize** F
4: **for** $i = 1$ to t **do**
5: $\quad X' \leftarrow sample(X, \psi)$ {without replacement}
6: $\quad (min,min) \leftarrow$ InitialiseWorkingSpace(X')
7: $\quad F \leftarrow F \cup$ SingleHS*-Tree$(X', min, max, 0)$
8: **end for**
9: **return** F

The aim is to generate many diverse HS*-Trees to form an ensemble. This is achieved by defining a (random) range for each dimension, forming a working space which covers

Algorithm 2 : SingleHS*-Tree(X, min, max, ℓ)

Inputs: X - input data, min & max - arrays of minimum and maximum values for all attributes in a working space, ℓ - current depth level

Output: an HS*-Tree

if ($|X| \leq SizeLimit$) or ($\ell \geq MaxDepthLimit$) **then**
  return *exNode*($Size \leftarrow |X|$)
else
  randomly select an attribute q
  $p \leftarrow (max_q + min_q)/2$
  $X_l \leftarrow filter(X, q < p)$
  $X_r \leftarrow filter(X, q \geq p)$
  {Build two nodes: $Left$ and $Right$ as a result of a split into two equal-volume half-spaces.}
  $temp \leftarrow max_q$; $max_q \leftarrow p$
  $Left \leftarrow$ SingleHS*-Tree($X_l, min, max, \ell + 1$)
  $max_q \leftarrow temp$; $min_q \leftarrow p$
  $Right \leftarrow$ SingleHS*-Tree($X_r, min, max, \ell + 1$)
  return *inNode*($Left, Right, SplitAtt \leftarrow q, SplitValue \leftarrow p$)
end if

all the training data of a subsample, before the construction of a tree. This is done in the step to **InitialiseWorkingSpace** in Algorithm 1. For each attribute q, a random split value (z_q) is chosen within the range [$Dmin_q$,$Dmax_q$], i.e., the minimum and maximum values of q in the subsample. Then, attribute q of the working space is defined having the range [min_q, max_q] = [$z_q - r$, $z_q + r$], where $r = 2 \cdot \max(z_q - Dmin_q, Dmax_q - z_q)$. The ranges for all dimensions define the working space to generate a Half-Space Tree. The outer rectangles in both Figures 4a and 4b are examples of a working space.

Constructing a single Half-Space Tree is almost identical to constructing an ordinary decision tree[4] [28], except that no splitting selection criterion is required at each node. The split point of a node in Half-Space Tree is simply the mid-point in a randomly selected dimension of the working space defined above. The detail procedure is described in Algorithm 2.

Note that the entire training procedure does not require any evaluation criteria at all; and randomisation is invoked in three steps: the random subsample step, the working space initialisation step, and the random attribute selection at each internal node.

Testing. During testing, a test instance x traverses through each Half-Space Tree from the root to an external node, and the mass recorded at the external node is used as the anomaly score $s(x)$ (i.e., Equation (2)) for this instance. This testing is carried out for all Half-Space Trees in the ensemble, and the final score is the average score from all trees, equivalent to Equation (1).

For a set of data set, the scores obtained are used to rank the instances. From this ranking and the ground truth, we employ AUC (Area Under receiver operating characteristic Curve) to measure the performance of all anomaly detectors reported in this paper.

4. However, they are for different tasks: Decision trees are for supervised learning tasks; Half-Sapce trees are for unsupervised learning tasks.

Time and Space complexities. Because of no evaluations or searches at all, an HS*-Tree can be generated very fast. In addition, a good performing HS*-Tree can be generated using only a small subsample (size ψ) from a given data set of size n, where $\psi \ll n$. An ensemble of HS*-Trees has training time complexity $O(th\psi)$ which is constant for an ensemble with fixed subsample size ψ, maximum depth level h and ensemble size t. It has time complexity $O(thn)$ during testing. The space complexity for HS*-Trees is $O(th\psi)$ and is also a constant for an ensemble with fixed subsample size, maximum depth level and ensemble size.

5 Empirical Evaluation

Here we conduct experiments to evaluate the proposed ranking measure: mass, and the proposed mass-based method HS*-Trees. This section has two aims. First, we compare the proposed ranking measure, mass, with two commonly used measures, density and distance, all implemented in HS*-Trees. This assesses the anomaly detection performance of the three measures using the same algorithm. Second, we assess the anomaly detection performance of mass-based method HS*-Trees in comparison with three state-of-the-art algorithms which uses density and distance ranking measures.

Experimental settings. The default settings for HS*-Trees are $\psi = 256$, $S = 20$, $h = 20$ and $t = 100$ (i.e., an ensemble of 100 trees is built.) The performance measures are AUC—Area Under receiver operating characteristics (ROC) Curve and CPU run time. Although AUC or ROC curve (or the alternative precision-recall curve) is commonly used for supervised learning tasks, it can also commonly used for measuring performance in anomaly detection, e.g., in [13, 16, 21]. AUC ranges from 0 (the worse) to 1 (the best). The results reported are an average over ten runs; each run is obtained using a different random seed for all non-deterministic algorithms. They are conducted as single threaded jobs processed at 2.3GHz in a Linux cluster (www.vpac.org).

When using kth NN distance or kNN density in HS*-Trees in the first experiment, a test instance traverses from the root of a tree to the deepest node which has at least k data points so that a value can be computed using this measure. We use $k = 5$ in our experiments.

The second experiment compares HS*-Trees with ORCA [8], one-class SVM (first mentioned in [31]) and LOF [9]. ORCA employs distance-based definition (ii), stated in section 3.1, to rank anomalies; LOF is the state-of-the-art density-based anomaly detector, designed to detect both local and global anomalies; it computes local density and defines anomalies as having low local densities (i.e., density-based definition (iv)); and SVM employs a simplified version of distance measure.

ORCA is a distance-based method based on k-Nearest Neighbour (kNN) with a sample randomisation scheme and a pruning rule to speed up run time. Our default parameters for ORCA are $k = 5$ and $N = \frac{n}{8}$, where N the number of anomalies expected. LOF is a density-based method based on

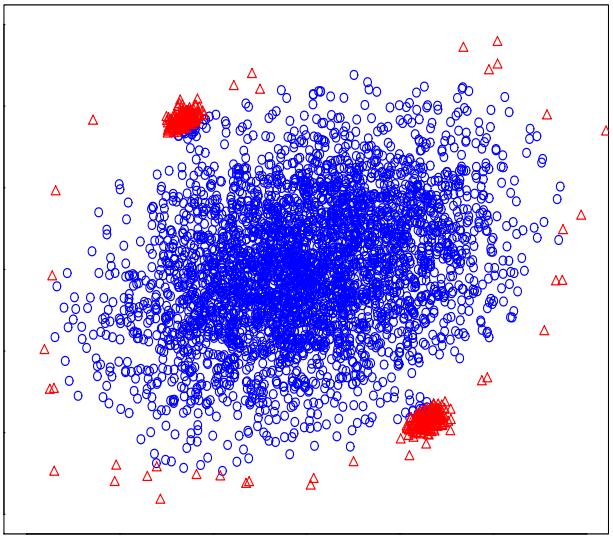


Fig. 6. This figure shows an example of Mulcross in two dimensions. Circles (○) are normal points in the middle cluster, and triangles (△) denote anomalies which are points as two smaller clusters and at the fringe of the middle cluster.

	data size	d	anomaly class
Http (KDDCUP99)	567497	3	attack (0.4%)
ForestCover	286048	10	class 4 (0.9%) vs. class 2
Mulcross	262144	4	2 clusters (10%)
Smtp (KDDCUP99)	95156	3	attack (0.03%)
Shuttle	49097	9	classes 2,3,5,6,7 (7%)
Mammography	11183	6	class 1 (2%)
Annthyroid	7200	6	classes 1, 2 (7%)
Satellite	6435	36	3 smallest classes (32%)
Pima	768	8	pos (35%)
Breastw	683	9	malignant (35%)
Arrhythmia	452	274	classes 03,04,05,07, 08,09,14,15 (15%)
Ionosphere	351	32	bad (36%)
Synthetic	2000	2	anomaly (7.5%)

TABLE 2
Data characteristics of the data sets used in the experiments, where d is the number of dimensions, and the percentage in bracket indicates the percentage of anomalies.

k-nearest neighbour. LOF's default parameter is $k = 10$. One-class SVM is using the Radial Basis Function kernel and its inverse width parameter is estimated by the method suggested in [10].

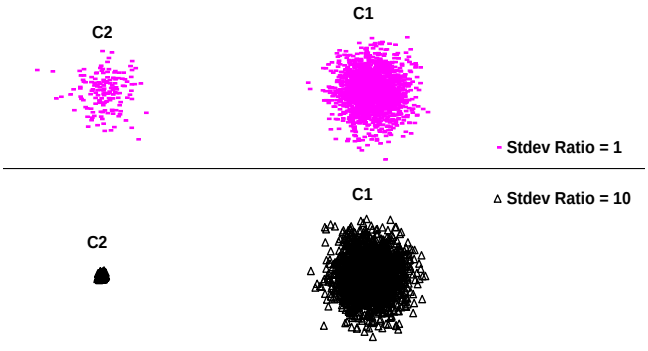
Benchmark data sets are employed in both expriments. We utilise a synthetic data and an additional of twelve data sets mostly from the UCI repository [5], which include many real-world data sets, e.g., two from KDD CUP 99, one Mammography data set [5] and one anomaly data generator Mulcross [30] which generates data with both clustered and scattered anomalies. An example of Mulcross is shown in Figure 6.

5. The Mammography data set was made available, courtesy of Aleksandar Lazarevic

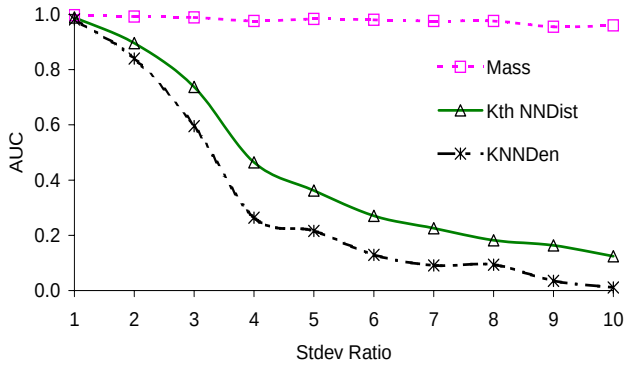
Table 2 gives a summary of these datasets. All nominal and binary attributes are removed to focus on the continuous-valued attributes.

5.1 Mass versus density and distance

To better understand the robustness of ranking using mass, we first examine in this section how well the ranking based on either mass, kth NN distance or kNN density in a scenario in which the density of an anomaly cluster changes with respect to the normal cluster. To achieve this, we use a synthetic data set with two clusters, C1 and C2. The normal cluster is denoted as C1: it is a bivariate normal distribution with a standard deviation of 2 and has 1850 instances. The anomaly cluster is denoted as C2: it has 150 instances and is well separated from the normal cluster. The standard deviation of C2 decreases from 2 to 0.2, with a step size of 0.2, yielding a ratio of standard deviations between C1 and C2 (denoted as Stdev Ratio), changing from 1 to 10—simulating an increasingly denser anomaly cluster. Figure 7(a) shows two examples where Stdev Ratio is equal to 1 and 10.



(a) Synthetic Data at StdevRatio=1 and StdevRatio=10



(b) Mass versus kth NN Distance and kNN density using HS*-Trees

Fig. 7. (a) Two examples of the synthetic data used. (b) A comparison of AUC performance of HS*-Trees using either mass, kth NN Distance or kNN density.

Figure 7(b) shows that when the standard deviation ratio changes from 1 to 10, the AUC score for HS*-Trees using either kth NN distance or kNN density drops significantly. When the ratio goes beyond 3, kNN density gives an AUC score below 0.5. Note that an AUC of 0.5 is the expected

	AUC			Time (Seconds)		
	mass	distance	density	mass	distance	density
Http	**1.00**	**1.00**	0.62	**88.35**	1849.21	568.33
ForestCover	**0.89**	0.65	0.67	**36.33**	156.20	131.48
Mulcross	**0.99**	#0.00	#0.00	**29.22**	182.71	109.18
Smtp	**0.90**	0.85	0.87	**17.07**	113.59	62.88
Shuttle	**1.00**	0.98	0.88	**9.75**	74.59	34.29
Mammography	**0.86**	0.82	0.83	**3.41**	11.51	7.00
Annthyroid	**0.73**	**0.73**	**0.73**	**2.61**	5.64	3.68
Satellite	**0.74**	**0.74**	0.72	**2.54**	6.37	4.94
Pima	0.69	**0.72**	**0.72**	1.44	1.68	**1.30**
Breastw	**0.99**	0.98	0.98	1.51	1.78	**1.26**
Arrhythmia	**0.84**	0.82	0.82	**1.92**	3.78	3.30
Ionosphere	0.80	**0.90**	**0.90**	1.36	1.59	**1.33**
win/draw/loss		7/3/2	9/1/2		12/0/0	9/0/3

TABLE 3
Result comparing three ranking measures in terms of AUC and runtime. The distance and density are computed based on k nearest neighbours at the leaf of HS*-Trees. Figures boldfaced are the best performance for each data set. The runtime results include both training and testing times. # The AUC results are not exactly zero but a small number less than one-hundredth.

score by *random* ranking. As the ratio increases, both kNN density and kth NN distance begin to rank normal instances ahead of normal instances—exactly the scenario depicted in (c) in Section 3. In contrast, the AUC score of HS*-Trees using mass has almost perfect score throughout the entire range.

We also conduct an experiment with k-Means using the three ranking measures and it produces the same result. This is shown in Appendix A—this shows that the mass ranking measure is better than either distance or density measure, independent of the specific method used (HS*-Trees or k-Means.)

It is important to note that while kth NN distance is able to detect the clustered anomalies less than k, trying to find an appropriate k is impractical for two reasons. First, there may be more than one cluster with differing numbers of anomalies. Second, the number of anomalies in one cluster can vary from one occassion to another, e.g., from one sample to another. Thus, setting a fixed k is not a solution. In addition, setting a large k increases the runtime substantially.

Table 3 shows the anomaly detection performance in the twelve data sets (listed in Table 2) in terms of AUC and runtime for HS*-Trees which uses each of the three measures to perform ranking. In terms of AUC, mass has more wins than losses than either distance or density, notably in ForestCover and Mulcross in which anomaly clusters have a significant presence. Only in the Pima and Ionosphere data sets mass loses, and the difference is small. These are likely to be due to scenario (d) mentioned in Section 3.2 in which both density and distance have a slight advantage. In terms of runtime, HS*-Trees using mass has a significant advantage over all other ranking measures, especially in large data sets. For example, in the largest data set Http, HS*-Trees using mass takes less than one-twentieth and one-sixth of the time required by HS*-Trees using distance and density, respectively.

5.2 Compare to state-of-the-art anomaly detectors

This experiment aims to show that mass-based HS*-Trees performs better, in terms of AUC and run time, than either distance-based or density-based methods: ORCA, SVM and LOF.

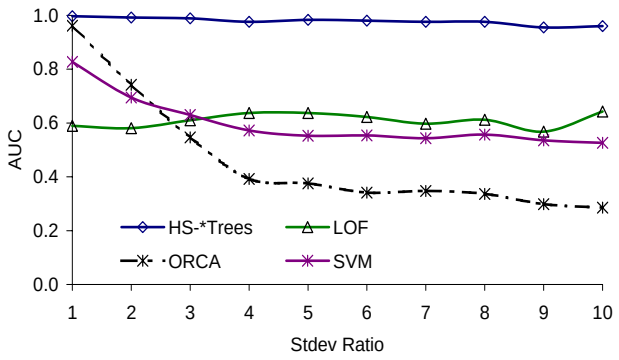


Fig. 8. A comparison of HS*-Trees, LOF, ORCA and SVM in the synthetic data.

Figure 8 shows the result using the synthetic data (described in Figure 7(a).) It shows that HS*-Trees has near perfect AUC over the entire range of Stddev Ratios. In contrast, methods based on distance and density all perform very poorly in detecting clustered anomalies in this data set, especially when clustered anomalies become significantly denser than the normal cluster. Note that LOF performs particularly poorly in this data set because it can only detect instances at the fringes of the two clusters as anomalies and that does not change much throughout the entire range of Stddev Ratios.

Table 4 shows the anomaly detection performance in the twelve data sets in terms of AUC and runtime, comparing HS*-Trees with the three methods: ORCA, SVM and LOF. In terms of AUC, HS*-Trees has a better detection accuracy than all of the other methods with the following win/draw/loss counts: 10/0/2 compared to ORCA, 11/0/1 compared to SVM, 10/0/1 compared to LOF. In terms of runtime, HS*-Trees is significantly faster than all other methods, especially in the large data sets. For example, in the largest data set Http, HS*-Trees takes less than one-hundredth and one-four-hundredth

	AUC				Time (Seconds)			
	HS*-Trees	ORCA	SVM	LOF	HS*-Trees	ORCA	SVM	LOF
Http	**1.00**	0.36	0.90	†	**88.35**	9487.47	35872.09	$>$ 2 weeks
ForestCover	0.89	0.83	**0.90**	0.57	**36.33**	6995.17	9737.81	224380.19
Mulcross	**0.99**	0.33	0.59	0.59	**29.22**	2512.20	7342.54	156044.13
Smtp	**0.90**	0.87	0.78	0.32	**17.07**	267.45	986.84	24280.65
Shuttle	**1.00**	0.60	0.79	0.55	**9.75**	156.66	332.09	7489.74
Mammography	**0.86**	0.77	0.65	0.67	**3.41**	4.49	10.80	14647.00
Annthyroid	**0.73**	0.68	0.63	0.72	2.61	**2.32**	4.18	72.02
Satellite	**0.74**	0.65	0.61	0.52	**2.54**	8.51	8.97	217.39
Pima	0.69	**0.71**	0.55	0.49	1.44	**0.06**	**0.06**	1.14
Breastw	**0.99**	0.98	0.66	0.37	1.51	**0.04**	0.07	1.77
Arrhythmia	**0.84**	0.78	0.71	0.73	1.92	0.49	**0.15**	6.35
Ionosphere	0.80	**0.92**	0.71	0.89	1.36	**0.04**	**0.04**	0.64
win/draw/loss		10/0/2	11/0/1	10/0/1		7/0/5	8/0/4	9/0/2

TABLE 4
Result comparing four anomaly detectors in terms of AUC and runtime. Figures boldfaced are the best performance for each data set. † We do not have the full results for LOF because it has a high computational complexity and is unable to complete the large data set in more than two weeks.

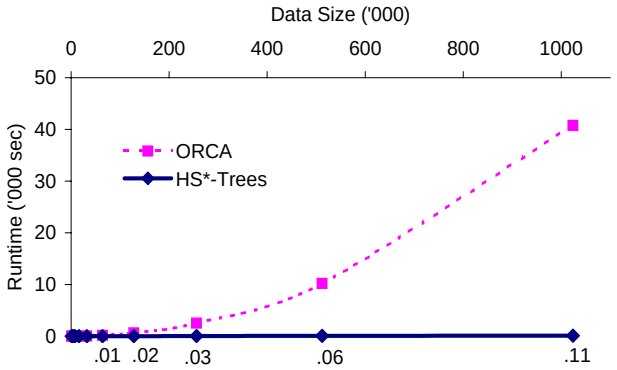


Fig. 9. Run time (training + testing) comparison between HS*-Trees and ORCA in Mulcross data set with increasing data sizes.

of the time required by ORCA and SVM, respectively; and the gap is even bigger in comparison with LOF—less than one-thirteen-thousandth!

There are two interesting observations. First, HS*-Trees using density (result shown in Table 3) performs better than LOF—a state-of-the art algorithm using density measure—in all data sets in terms of both AUC and runtime. The only exceptions are in the small size Pima and Ionosphere data sets in terms of runtime; and in Mulcross in terms of AUC. Second, HS*-Trees using distance performs better in terms of AUC than ORCA (which employs distance measure) in 7 data sets, draws in 1, and loses in 4 data sets, which is very competitive. In terms of runtime, HS*-Trees using distance is significantly faster in all large data sets—it is more than five times faster than ORCA in the largest data set, Http.

Figure 9 shows the runtime comparison between HS*-Trees and ORCA (which is a near-linear algorithm and the fastest among the three anomaly detectors) using the Mulcross data generator.

In this experiment, training an ensemble of 100 HS*-Trees takes constant time at about 1 second, no matter the given data size is one thousand or one million. This is because only a subsample of fixed size $\psi = 256$ is used to train an HS*-Tree. The sublinear increase in runtime for HS*-Trees is solely due to the time used for testing, which is sublinear to the size of the data set. In this case, the testing time increases from 0.6 to 111 seconds when the data size increases from one thousand to one million. In contrast, ORCA's runtime increases from 0.1 seconds to over 40000 seconds. Moreover, the AUC performance of HS*-Trees has already reached 0.97 from one thousand data size; whereas ORCA's AUC starts at 0.02 and stabilises at 0.33, as the top AUC performance, even the data size increases to one million.

6 Time and Space Complexities Comparison.

This section compares the time and space complexities of three state-of-the-art anomaly detectors with HS*-Trees. This includes DOLPHIN [3] — the latest k-nearest neighbour-based algorithm method which uses distance for ranking. Table 5 lists the complexities for the four algorithms.

	Time complexity	Space complexity
HS*-Trees	$O(th(n+\psi))$	$O(th\psi)$
DOLPHIN	$O(n^2d)$	$O(n)$
	$O(\frac{k}{p}nd)^\dagger$	$O(\frac{k}{p})^\dagger$
ORCA	$O(nlogn \cdot d)$	$O(n)$
LOF	$O(n^2d)$	$O(n)$

† Under special condition; p is the probability of randomly picking a point from the data set which is a neighbour of the point under consideration using a search index; k is the number of nearest neighbours; d is the number of dimensions.

TABLE 5
A comparison of time and space complexities. The time complexity includes both training and testing.

HS*-Trees has a significant advantage over three k-nearest-neighbours-based methods in terms of both time and space complexities. This is mainly due to the fact that HS*-Trees only needs a small subsample to train a tree, where $\psi \ll n$; ψ (also t and h) can be fixed in practice, regardless of the size

of the given training set, as demonstrated in the last section using Mulcross. Another distinguishing feature is that the time complexity of HS*-Trees is independent of the dimensionality of the domain. Thus, the training time and memory space requirement are fixed—these properties make HS*-Trees the ideal candidate to apply to domains with huge data size or infinite data such as data stream.

7 Relation to iForest

Here we establish that the anomaly score used in iForest [20], i.e., path length, is a form of the augmented mass in Half-Space Tree.

The definition of anomalies based on iForest is given as follows.

'Anomalies are the top-ranked instances whose average path lengths are the shortest.'

During testing, a tree in iForest computes the path length at an external node with m instances at level ℓ as follows.

$$\begin{aligned} s &= \ell + c(m) \\ &= \ell + 2(ln(m-1) - \frac{m-1}{m} + E), \end{aligned}$$

where $c(m)$ is a function which estimates the average path length of an unexpanded subtree for a training data of size m, and E is Euler's constant [20].

Apply logarithm to Equation (2) used in Half-Space Tree gives

$$s' = \ell + log(m)$$

Note that both s and s' take the general form of depth level ℓ plus a derivative of mass. In essence, the path length used in iForest [20] is a form of augmented mass ranking measure. Thus, iForest is a kind of mass-based approach.

Another key difference between Half-Space Tree and iForest [20] is that Half-Space Tree uses the mid-point split which guarantees equal-size subdivision; whereas iForest randomly selects a split point. The analysis in Section 4.1 is possible because of half-space splits; as far as we know, there is no equivalent analysis exists for random-split.

A comparison of HS*-Trees and iForest in terms of AUC is given in Appendix B.

8 Related work

On the surface, there is a close relationship between the augmented mass in Equation (2) and data depth [19] because of the use of depth level in the form of Half-Space Tree: they both delineate the centrality of a data cloud (as opposed to compactness in the case of density measure.) However, there are two fundamental differences. First, mass, by its definition, does not have to be expressed in tree form. For example, using a clustering algorithm to find regions, and then apply mass to rank the regions (as shown in Appendix A) does not entail the concept of depth or centrality in each region. Second, mass is a simple and straightforward measure; whereas data depth has many different definitions, depending on the construct used to define depth. The constructs could be Convex Hull, simplicial, half-space[6] and so on [19], all of which are expensive to compute in multi-dimensional problems. Even when expressed in the form of Half-Space Tree, the augmented mass is still simple and straightforward and can be traced back to the simple mass definition as shown in Section 4.2.

At the algorithm level, k-d Tree [17], that based on median split, may appear to be similar to Half-Space Tree on the surface. However, there are important differences. First, the purpose of the algorithm is different: k-d Tree is designed to speed up search, e.g., in a near neighbour search; whereas Half-Space Tree is specifically designed for mass estimation (the new ranking measure we proposed here) for the purpose of anomaly detection. Second, constructing a node of a k-d Tree starts by searching for median as the splitting point on one dimension, and it cycles through the dimensions to build subsequent nodes in the tree; in contrast, the splitting point for a node of Half-Space Tree is simply the mid-point of a randomly chosen dimension in the working space, independent of the distribution of the data—no search is required to find the split point. Third, a k-d Tree cannot be used to estimate mass because it is a balance tree (due to the median-split) and all external nodes will have the same mass—useless for our purpose here! Fourth, a k-d Tree is constructed using all available data; whereas each Half-Space Tree only requires a small training sample; e.g., only 0.045% or 256 out of more than half a million instances in the Http data (reported in Section 5) are required to build a good-performing Half-Space Tree. Although both are linear time-complexity algorithms in training, k-d Tree is linear with respect to the total training set size n; and Half-Space Tree is linear with repsect to $\psi \ll n$.

A method has been proposed to use k-d Tree for anomaly detection [12]. It partitions the data into regions of uniform density and then ranks the regions according to their densities. It will have problems detecting clustered anomalies because of the use of density measure to do ranking.

LOCI [26] is a density-based method that uses mainly countings to compute its anomaly score because density (= number of instances per unit space) is equivalent to counts when the space is the same for all density computations. For each instance p, it first identifies a region defined by a (fixed user-defined) radius r from p and all its nearest neighbours within the region. Then, it counts the number of instances within a smaller circle of radius αr, centred at each nearest neighbour including p, where $\alpha < 1$. The anomaly score, Multi-Granularity Deviation Factor (MDEF), a derivative of density ranking measure, is defined to be the relative difference between the average count for all nearest neighbours ($\hat{n}$) and the count for p (n), i.e., $(\hat{n} - n)/\hat{n}$. A point which is surrounded by points having the same density will have MDEF=0. Anomalies will have MDEF much larger than 0. It is a more computational intensive approach than Half-Space Tree as it requires distance calculation to define the regions.

Tietjen and Moore [33] describe 'masking effect' of clustered anomalies as follows:

6. The term 'half-space' has been used in geometry to denote either part of the space divided by a hyperplane; they are not required to have equal-size space, unlike the one used in Half-Space Tree.

"Suspected observations sometimes form subgroups; i.e., several values are closer to each other than they are to the bulk of the observations. .... The masking effect is the inability of a testing procedure to identify even a single outlier in the presence of several suspected values."

Statistical tests are all affected by the masking effect in different degrees [e.g., 7,27]. For example, in the flow rate data set which has four anomalies clusters making up 20% of the total 2589 hourly average flow rate measurements from an industrial manufacturing process, Pearson [27, Chapter 3] shows that the Hampter identifier [14] is better anomaly detector than extreme studentised deviation identifier [14]; yet in another data set which has asymmetrical distribution, the Hampter identifier misses some scattered anomaly while the asymmetrical boxplot is able to detect it together with other anomalies. All of these statistical tests are designed for single or low dimensional problems only [11].

Note that the 'collective outliers' referred to in [11] are different from clustered anomalies. Collective outliers occur in data where individual instances are related, in e.g., sequence, spatial, graph and time series data. The individual instances in collective outliers may not be anomalies by themselves; whereas every individual instance in clustered anomalies is an anomaly, without exception.

9 Concluding Remarks

This paper introduces a ranking measure, mass, for anomaly detection. This measure is simple, straightforward and fast to compute. It is the only basic ranking measure that we know which ranks both scattered and clustered anomalies correctly for anomaly detection tasks.

We have identified the key weakness for the two commonly used measures: distance and density—as a ranking measure, they both fail to rank clustered anomalies correctly; thus unable to detect this kind of anomalies. It is well-known that both of these measures are computationally expensive. Existing anomaly detection methods based on them require extensive search and pruning heuristics in order to speed up the run time (e.g., [3,8]).

Our analysis using Half-Space Tree shows that mass is an effective ranking measure for both scattered anomalies and clustered anomalies, and the depth level of the tree is a proxy to mass. This simple method is shown to perform better than state-of-the-art distance-based anomaly detectors ORCA and SVM, and density-based anomaly detector LOF, in terms of anomaly detection accuracy. Its time and space complexities are also significantly better with constant training time and memory space requirement.

We reveal that a previous method iForest is a mass-based approach which employs the depth level as a proxy to mass. This uncovers the principle underpinning the method, which was previously unknown.

This paper identifies the source of failure for existing methods to detect clustered anomalies—the use of density or distance as the ranking measure. There are a few attempts to mitigate the problem with limited success, e.g., modifying the density measure [32] or alternatively use a clustering algorithm to identify the clustered anomalies. We show that by using the mass measure and the proposed HS*-Trees, clustered anomalies can be identified effectively and efficient without resorting to the use of clustering algorithm.

References

1) Abe, N., Zadrozny, B., and Langford, J. (2006). Outlier detection by active learning. Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 504–509.
2) Aleskerov, E., Freisleben, B., and Rao, B. (1997). Cardwatch: A neural network based database mining system for credit card fraud detection. Proceedings of IEEE Computational Intelligence for Financial Engineering. 220–226.
3) Angiulli F., F. Fassetti. (2009). DOLPHIN: An efficient algorithm for mining distance-based outliers in very large datasets. ACM Transactions on Knowledge Discovery from Data. Vol 3 Issue 1. Article No.4.
4) Angiulli, F. and C. Pizzuti. (2002). Fast outlier detection in high dimensional spaces. In Proceedings of the Sixth European Conference on the Principles of Data Mining and Knowledge Discovery. 15–26.
5) A. Asuncion and D. Newman. UCI machine learning repository. 2007.
6) Barnett, V. (1976). The ordering of multivariate data. Journal of the Royal Statistical Society. Series A 139, 318–354.
7) Barnett, V. and Lewis, T. (1994). Outliers in statistical data. John Wiley and sons.
8) Bay, S. D. and Schwabacher, M. (2003). Mining distance-based outliers in near linear time with randomization and a simple pruning rule. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. 29–38.
9) Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof: identifying density-based local outliers. In Proceedings of 2000 ACM SIGMOD International Conference on Management of Data. 93-104.
10) B. Caputo, K. Sim, F. Furesjo, and A. Smola. (2002). Appearance based object recognition using svms: which kernel should I use? Proc of NIPS workshop on Statitsical methods for computational experiments in visual processing and computer vision.
11) Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly Detection : A Survey. ACM Computing Surveys, Vol. 41(3). 1–58.
12) Chaudhary, A., Szalay, A. S., and Moore, A. W. (2002). Very fast outlier detection in large multidimensional data sets. Proceedings of ACM SIGMOD Workshop in Research Issues in Data Mining and Knowledge Discovery (DMKD). ACM Press.
13) Das, K., Schneider, J. and Neill, D.B. (2008). Anomaly Pattern Detection in categorical datasets. Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining. 169–176. ACM Press.

14) Davies, L., and Gather, U. (1993). The identification of multiple outliers. Journal of the American Statistical Association. 88. No.423. 782–792.
15) Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P.-N., Kumar, V., Srivastava, J., and Dokas, P. (2004). MINDS - Minnesota Intrusion Detection System. In Data Mining - Next Generation Challenges and Future Directions. MIT Press.
16) Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. In Proceedings of the Seventeenth International Conference on Machine Learning. 255–262.
17) Friedman, J.H., Bentley, J.L., and Finkel, R.A. (1997). An Algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software 3(3): 209–266.
18) Hawkins, D. (1980). Identification of outliers. Monographs on Applied Probability and Statistics.
19) Liu, R.Y., J. M. Parelius, K. Singh. (1999). Multivariate Analysis by Data Depth: Descriptive Statistics, Graphics and Inference. The Annals of Statistics. Vol. 27, No.3. 783–840.
20) Liu, T., Ting, K.M. & Zhou, Z-H. (2008). Isolation Forests. Proceedings of the 2008 IEEE International Conference on Data Mining. pp. 413-422.
21) Kriegel, H-P., Schubert, M. & Zimek, A. (2008). Angle-based outlier detection in high-dimensional data. Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. 444-452.
22) Knorr E. M. and R. T. Ng. (1999). Finding intensional knowledge of distance-based outliers. In Proceedings of the 25th VLDB Conference. 211–222
23) Knorr, E. M., Ng, R. T., and Tucakov, V. 2000. Distance-based outliers: algorithms and applications. The VLDB Journal 8, 3-4, 237–253.
24) MacQueen, J.B. (1967). Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. 281–297.
25) Murphy, R.B. (1951). On Tests for Outlying Observations. PhD thesis, Princeton University.
26) Papadimitriou, S. Kitagawa, H. Gibbons, P.B. Faloutsos, C. (2003). LOCI: fast outlier detection using the local correlation integral. Proceedings of 19th International Conference on Data Engineering. pp. 315-326. IEEE Computer Society.
27) Pearson, R. (2005). Mining imperfect data : dealing with contamination and incomplete records. Philadelphia : Society for Industrial and Applied Mathematics.
28) Quinlan, J.R. (1993). C4.5: programs for machine learning. Morgan Kaufmann.
29) Ramaswamy, S., Rastogi, R., and Shim, K. 2000. Efficient algorithms for mining outliers from large data sets. Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 427–438.
30) Rocke D. M. and D. L. Woodruff. (1996). Identification of outliers in multivariate data. Journal of the American Statistical Association. 91(435):1047-1061.
31) Schoelkopf, B., R. C. Williamson, A. J. Smola, J. Shawe-Taylor and J. C. Platt (2000). Support vector method for novelty detection, Advances in Neural Information Processing Systems, vol. 12, pp. 582-588.
32) Tang, J., Chen, Z., Fu, A.W.-c. and Cheung, D.W. (2002). Enhancing effectiveness of outlier detections for low density patterns. Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 535–548.
33) Tietjen, G.L. and Moore, R.H. (1972). Some Grubbs-Type Statistics for the Detection of Several Outliers. Technometrics. 14. 583–597.
34) Wong, W.-K., Moore, A., Cooper, G., and Wagner, M. (2003). Bayesian network anomaly pattern detection for disease outbreaks. Proceedings of the 20th International Conference on Machine Learning. AAAI Press, Menlo Park, California, 808–815.

APPENDIX A: ANOMALY DETECTION USING CLUSTERING.

Here we demonstrate that one can use a clustering algorithm to carve out the regions in the feature space, and then use any one of the three ranking measures to do ranking for anomaly detection. We employ a commonly used clustering algorithm k-Means [24] to carve out a pre-set number of clusters from data.

We perform the same experiment as conducted in Section 5.1. We compare the ranking performance using either one of the three ranking measures after the k-Means clustering result in the synthetic data set (with increasing densities for the anomaly cluster.)

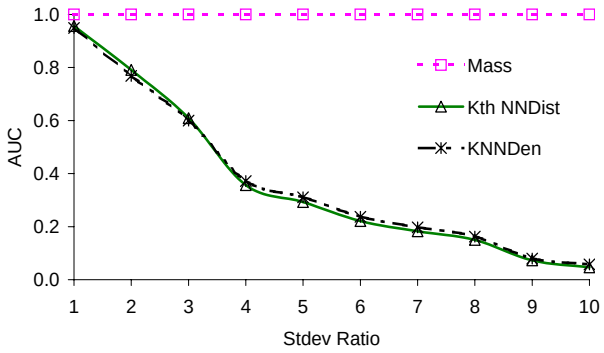


Fig. 10. A comparison of AUC performance of k-means (k=5) using either mass, kth NN distance or kNN density.

Figure 10 shows the result comparing the three measures using the synthetic data set. It shows the same relative performance between mass and the other two ranking measures, as we have seen in Section 5.1—mass is a better ranking measure no matter HS*-Trees or k-means is used to carve out the regions.

APPENDIX B: HS*-TREES VERSUS IFOREST

This section compares the anomaly detection performance between HS*-Trees and iForest [20]. Table 6 shows that HS*-Trees performs better than iForest in 6 data sets, draws in 4, and loses only in 2 data sets, in terms of AUC.

	HS*-Trees	iForest
Http	**1.00**	**1.00**
ForestCover	**0.89**	0.88
Mulcross	**0.99**	0.97
Smtp	**0.90**	0.88
Shuttle	**1.00**	**1.00**
Mammography	**0.86**	**0.86**
Annthyroid	0.73	**0.82**
Satellite	**0.74**	0.71
Pima	**0.69**	0.67
Breastw	**0.99**	**0.99**
Arrhythmia	**0.84**	0.80
Ionosphere	0.80	**0.85**

TABLE 6
AUC result comparing HS*Trees and iForest. Figures boldfaced are the best performance for each data set.

Gippsland School of Information Technology

Technical Report

Title: Streaming HS-Trees: Amortised O(1) anomaly detection algorithm in evolving data streams

Authors: Swee Chuan Tan, Kai Ming Ting, Fei Tony Liu

TR No.: TR2009/2

Department: Gippsland School of Information Technology

Faculty: Information Technology

This page is intentionally left blank

Streaming HS-Trees: Amortised $O(1)$ anomaly detection algorithm in evolving data streams

Swee Chuan Tan, Kai Ming Ting and Fei Tony Liu*

Abstract

The data stream problem has received a lot of attention in recent years. Data streams are potentially high speed and infinite, demanding efficient algorithms that require only one pass over the data. Furthermore, a lot of stream data can evolve over time, thus data stream algorithms must be able to adapt to changes occur at the input. In this paper, we propose a new, adaptive, data stream anomaly detection method, called Streaming HS-Trees. The method features a *random tree model* well integrated with a *change detector*, so that the model can adapt to changing input. We also identify the specific type of change in data distribution that requires model update in order to maintain high detection accuracy. The proposed model can be constructed without any data instances and hence can be installed before the arrival of a data stream. It is highly efficient because it requires no model restructuring when adapting to new data distribution. Our analysis shows that Streaming HS-Trees has an amortised constant time complexity and has a constant memory requirement, independent of data size. Our performance study demonstrates the benefits of developing an anomaly detection model that can adapt to data distribution changes. When compared with two existing outlier detection methods, our method performs favourably in terms of both detection accuracy and runtime performance.

1 Introduction

Data streams are commonly found in modern data acquisition systems. Sensor networks, for example, generate vast amount of data that must be analysed in real time. Data streams are potentially infinite. Any off-line learning algorithms that attempt to store all the data for analysis will run out of memory space at some point, regardless of the capacity of the available memory space.

In addition, no single static model can accurately analyse an entire data stream that evolves and experiences changes in data distribution over time. Instead, the model needs to adapt to different parts of the data stream.

A data stream algorithm thus need to satisfy two constraints. First, it must be a one-pass algorithm, i.e., inspect each data point only once—it discards a data point before the next is processed. In a one-pass algorithm, the memory requirement never grows, and it can analyse potentially endless amount of data. Second, it must incorporate change detection and model update mechanisms into the method, in order to deal with time-varying data distribution.

This paper proposes a one-pass anomaly detector for evolving data streams. The proposed method is called Streaming Half-Space Trees (HS-Trees).

Streaming HS-Trees has the following characteristics:

i) It is a one-pass anomaly detection algorithm that can be used to analyse massive datasets or data streams.

ii) Unlike traditional model-based algorithms, a model in Streaming HS-Trees can be built without any data; hence it is possible to create HS-Trees before the arrival of a data stream.

iii) Each model stores the profile of a data stream in different time windows. As will be shown later, the profiles between two windows can be compared easily, offering a simple way to detect distribution changes in data stream.

iv) When there is a need to update a model, Streaming HS-Trees does so by simply using the latest profile in the latest time window; this avoids the need and the cost to modify its model structure.

v) It has the ability to deal with anomaly detection and data distribution change within a single framework.

The rest of this paper is organized as follows. Section 2 states the problem and the goal of this paper. Section 3 presents the proposed Streaming HS-Trees method. Section 4 describes the experimental setup and Section 5 discusses the experimental results. Section 6 provides a discussion. Finally, we conclude this paper in Section 7.

2 Problem Statement and Goal

When a data stream arrives continuously in high speed, it is impractical to use traditional off-line learning methods that store all the data for analysis. Instead, an on-line, one-pass anomaly detector is required to address this problem.

The underlying profile in a stream may change over time, causing any non-adaptive anomaly detector to degrade its detection accuracy. However, only a certain type of change demands a model update; whereas no model update

*Gippsland School of Information Technology, Monash University, Australia.

is required in others. Thus, we need a change detector that is able to detect the 'right' type of change and then trigger the anomaly detector to adapt to the new profile.

A data distribution can be expressed as $P(x, y)$ which has two components: $P(x|y)$ and $P(y)$, where x is the input and y is the 'class'. In the anomaly detection context, $y \in \{normal, anomaly\}$. We describe the different types of change in data distribution as follows:

(I) Change due to normal points only: $P(x|y = normal)$

(II) Change due to anomalies only: $P(x|y = anomaly)$

(III) Change in proportion of anomalies/normal points only: $P(y)$, especially an increased number of anomalies or contamination level: $P(y = anomaly)$

In practice, a change may involve any combination of the above: $P(x, y) = P(x|y)P(y)$.

A model shall be updated for any combination of changes involving Type (I) change. However, it is imperative not to update a model when it is either Type (II) or Type (III) change. Type (I) and Type (II) changes are collectively referred to as concept change in the literature. We make a distinction here because the change in $P(x|y = anomaly)$ does not affect the detection performance of an anomaly detector if it profiles normal points. In fact, any attempt to update the model will result in a poor detection accuracy under this situation.

Problem statement. We address three inter-related problems in evolving data streams. The first problem is to maintain a high anomaly detection accuracy at all times in evolving data streams. This requires an anomaly detector to (i) have an ability to *detect change* in data distribution and (ii) effect a timely *model update* in order to adapt to the changing data distribution. As we have mentioned earlier that not all changes in data distribution warrant a model update. Thus, the second problem is to devise a change detection mechanism which will detect any change due to $P(x|y = normal)$ only; and ignore all other changes. The third problem is to institute a model update only when a persistent change in $P(x|y = normal)$ is detected; and no model update if the change is found to be transient.

Goal. Our goal in this paper is to address these three problems in a single framework. We aim to:

- introduce an algorithm that has an amortised $O(1)$ time complexity, called Streaming HS-Trees,
- demonstrate that Streaming HS-Trees can deal with anomaly detection, change detection, and adapting to data distribution change, all within a single framework.

3 The Proposed Method

The proposed method is a tree-based ensemble approach. The model is in the form of Half-Space Trees, or HS-Trees. We describe the proposed method in the following four sections. Section 3.1 provides the definitions and algorithms to construct an HS-Tree. Section 3.2 describes the proposed one-pass algorithm, Streaming HS-Trees. Section 3.3 provides an analysis of the time and space complexities of Streaming HS-Trees. Section 3.4 presents the change detection and model update mechanisms.

The key symbols and notations used in this paper are listed in Table 1.

x	a streaming point
n	the number of streaming points
T	an Half Space Tree, HS-tree
N	a node in an HS-Tree or $Node$
t	the number of HS-Trees in an ensemble
h	maximum depth level of a tree, or $maxDepth$
min	an array of minimum values for all dimensions
min_q	minimum value of dimension q
max	an array of maximum values for all dimensions
max_q	maximum value of dimension q
r	mass* of a node in the reference window
l	mass of a node in the latest window
ψ	window size
λ	number of consecutive windows before a model is updated
s	an anomaly score

Table 1: Key symbols and notations (*mass is the number of training instances which traverses through a node in an HS-Tree.)

3.1 Half-Space Trees. A Half-Space Tree is a balanced binary tree in which each internal node splits a feature (sub)space into two half-sized subspaces; and all external nodes have the same depth. An Half-Space Tree is built from a working space that is established by defining a range in every dimension. The tree building process begins by randomly selecting a dimension and then bisecting the working space using mid-point of the selected dimension. This process continues for each newly created node recursively until it reaches the required maximum depth level, denoted as h or $maxDepth$.

Creating diverse HS-Trees is crucial to the success of our method. This is achieved by invoking the procedure **InitialiseWorkingSpace** (Algorithm 1) to obtain a new working space, right before the construction of *each tree*. The purpose of Algorithm 1 is to redefine the range of each dimension in the feature space such that each dimension has a new range. Since each tree is built from a variant of the original space, the result is an ensemble of diverse HS-Trees.

Algorithm 2 shows the procedure for building a single HS-Tree. Each internal node is formed by randomly select-

Algorithm 1 : InitialiseWorkingSpace($Dmin, Dmax$)

Inputs: $Dmin$ & $Dmax$ - arrays of given minimum and maximum values for every dimension

Output: min & max - arrays of redefined minimum and maximum values for every dimension in a Working Space

1: **for** each dimension q **do**
2: Randomly choose s from $[Dmin_q, Dmax_q]$
3: $\sigma \leftarrow 2 \cdot \max(s - Dmin_q, Dmax_q - s)$
4: $min_q \leftarrow s - \sigma$
5: $max_q \leftarrow s + \sigma$
6: **end for**
7: **return** min and max

ing a dimension (see line 4) to form two half-spaces; the split point is the mid-point of the current range of the selected dimension. As a result, the entire tree construction procedure is very fast because the process requires no evaluation criteria for dimension or split point selections.

Each node has two mass variables, r and l, which record the number of training instances traversing through it at different time windows of a data stream. Each of these variables is assigned an initial value of zero during the initial tree construction process.

Algorithm 2 : BuildSingleHS-Tree(min, max, k)

Inputs: min & max - arrays of minimum and maximum values for every dimension in a Working Space, k - current depth level

Output: an HS-Tree

1: **if** $k == maxDepth$ **then**
2: return *Node*($r \leftarrow 0, l \leftarrow 0$) {External node}
3: **else**
4: randomly select a dimension q
5: $p \leftarrow (max_q + min_q)/2$
6: {Build two nodes: $Left$ and $Right$ as a result of a split into two equal-volume half-spaces.}
7: $temp \leftarrow max_q$; $max_q \leftarrow p$
8: $Left \leftarrow$ BuildSingleHS-Tree($min, max, k+1$)
9: $max_q \leftarrow temp$; $min_q \leftarrow p$
10: $Right \leftarrow$ BuildSingleHS-Tree($min, max, k+1$)
11: return *Node*($Left, Right, SplitAtt \leftarrow q,$ $SplitValue \leftarrow p, r \leftarrow 0, l \leftarrow 0$)
12: **end if**

Recording mass profile in HS-Tree. Once a HS-Tree is constructed, it needs to build a mass profile of the data before it can be employed for anomaly detection. The process involves traversing every training instance through the HS-Tree. The mass profile of a node is simply the number of training instances traversing through that node. Algorithm 3 shows that training instances in the reference time window will update mass r, otherwise mass l (in the latest time window) is updated. The variables r and l are used in Streaming HS-Trees which will be described in Section 3.2.

Algorithm 3 : UpdateMass($x, Node,$ *referenceWindow*)

Inputs: x - a test instance, $Node$ - a node in an HS-Tree

Output: none

1: (*referenceWindow*)? $Node.r$++ : $Node.l$++
2: **if** ($Node.Level < maxDepth$) **then**
3: Let $Node'$ be the sub-node of $Node$ that x traverses
4: UpdateMass($x, Node',$ *referenceWindow*)
5: **end if**

Scoring. After a mass profile is recorded in an HS-Tree, it is ready to assign anomaly scores to test instances. Given a test instance x and an ensemble of HS-Trees, the anomaly score is the sum of $score(x, T.root)$ (see Algorithm 4) obtained from every Half-Space Tree T.

In Algorithm 4, $sizeLimit$ is a global parameter that ensures that a score is obtained from a node that has a reasonable number of instances. $maxDepth$ is also a global parameter; it ensures that a tree is grown to the specified maximum depth. For our purpose, we find that $sizeLimit = 20$ and $maxDepth = 15$ give reasonable results over a range of real and synthetic datasets.

Algorithm 4 : Score($x, Node$)

Inputs: x - a test instance, $Node$ - a node in an HS-Tree

Output: an anomaly score for x

1: **if** ($Node.Level == maxDepth$) $\vee$ ($Node.r \leq sizeLimit$) **then**
2: return $Node.r \times 2^{Node.Level}$
3: **else**
4: Let $Node'$ be the sub-node of $Node$ that x traverses
5: Score($x, Node'$)
6: **end if**

3.2 Streaming HS-Trees. One interesting aspect of the proposed streaming method is that it requires no data to build a model—it only requires the range of each dimension in the feature space. This is materially different from many traditional models, which demand training data to construct their models. As a result, a model can be constructed with Streaming HS-Trees even before the actual data arrive, as long as the range of each dimension is known (or estimated) *a priori*. Furthermore, traditional algorithms must update their model structures continuously in order to keep up with newly arrived data. In contrast, Streaming HS-Trees builds a model structure that does not need to be changed to keep up with newly arrived data.

Algorithm 5 shows the operational procedure for

Streaming HS-Trees. Line 1 builds an ensemble of Half-Space Trees. Line 2 uses the first ψ instances of the stream to train the HS-Trees. Since these instances come from the initial reference time window, only mass r of each traversed node is updated. After these two steps, the model is ready to provide an anomaly score for each subsequent streaming point.

Streaming HS-Trees is designed to process the data in a single pass. When a data point arrives, it is traversed from the root to a (terminating) node of each tree. The point is then discarded before the next data point is processed. This enables the model to deal with an incoming data stream by examining every data point only once. Hence, the proposed method only requires a finite memory to process an infinitely long data stream.

Each node in an HS-Tree has two variables: reference mass r and latest mass l, which respectively store the mass profiles of a *reference* window and *latest* window. Line 2 of the algorithm records the mass profile in the reference mass r. This mass is always used to compute the anomaly score for each streaming point (line 9). The recording of mass for each subsequent streaming point in the latest window is carried out on mass l (line 8). This latest mass profile is used by the change detector to decide if a change (with respect to mass r in the reference window) has effected. Each node with a non-zero mass l is reset at the end of each window (line 19).

In Streaming HS-Trees, the model is updated only after a (persistent) distribution change is detected over λ consecutive windows (line 15). This is to avoid model update over a transient distribution change. Model update is surprisingly simple and no structural change of the model is required. The model is updated to the latest mass before the start of the next window by simply transferring the non-zero mass l to r (line 16).

3.3 Time and space complexities. Here, we analyse the **amortised time complexity** of Streaming HS-Trees. For n streaming points, where $n > \psi$, there are n predictions, $\frac{n}{\psi}$ change detections, and at most $\frac{n}{\lambda\psi}$ model updates. There are five key operations in the main loop of Algorithm 5, as listed in Table 2.

The first operation is to update the mass variable l in all nodes along a path from the root of a tree to the maximum depth of h; and this is required for all t trees. The second operation is to provide a score which takes $O(h)$ for each tree and a sum of scores over t trees. These two operations are to be carried out for each streaming point. The third operation is to detect change (the details are provided in Section 3.4) which needs to access a maximum of ψ nodes in each tree; and this operation is performed at the end of each window. The fourth operation is to update the model to the lastest mass profile which takes $O(\psi)$ number of assignments $r \leftarrow l$

Algorithm 5 : Streaming HS-Trees(ψ, t)

Inputs: ψ - Window Size, t - number of HS-Trees
Output: s - anomaly score for each streaming point

1: Build t HS-Trees : call Algorithms 1 & 2 for each tree
2: Record a reference mass profile in HS-Trees: for each tree T, invoke UpdateMass($x, T.root$, ***true***) for each item x in the first ψ instances of the stream
3: $Count \leftarrow 0$
4: **while** data stream continues **do**
5: Receive the next streaming point x
6: $s \leftarrow 0$
7: **for** each tree T in HS-Trees **do**
8: UpdateMass($x, T.root$, ***false***) {update mass l in T}
9: $s \leftarrow s +$ Score($x, T.root$) {accumulate scores}
10: **end for**
11: Report s as the anomaly score for x
12: $Count$++
13: **if** $Count == \psi$ **then**
14: $ChangeDetected$? $\#Change$++ : $\#Change \leftarrow 0$
15: **if** $\#Change \geq \lambda$ **then**
16: Update model : $N.r \leftarrow N.l$ for each non-zero mass node N
17: $\#Change \leftarrow 0$
18: **end if**
19: Reset $N.l \leftarrow 0$ for all non-zero mass node N
20: $Count \leftarrow 0$
21: **end if**
22: **end while**

for all non-zero mass nodes. The operation only needs to be done at most $\frac{n}{\lambda\psi}$ times. The fifth operation is to reset $N.l$ for all nodes N with non-zero mass. This operation involves a maximum of ψ non-zero nodes; it is done for every tree and needs to be carried out at the end of each window.

In summary, model update has the smallest cost among the five operations, while update mass l and score($x, T.root$) have the largest cost.

Thus, for Streaming HS-Trees, the average time cost of each operation in the worse case for n streaming points is

$$\frac{\mathcal{T}(n)}{n} = O(t(h + 1 + \frac{1}{2\lambda})).$$

Note that the above amortised cost is independent of data size n or the window size ψ or the number of dimensions. Since the parameters t, h and λ are algorithmic parameters independent of data size, our proposed algorithm Streaming HS-Trees is an amortised $O(1)$ algorithm.

The **space complexity** for HS-Trees is $O(t2^h)$ and is a constant for an ensemble with fixed maximum depth level (h) and ensemble size (t). Note that this is also independent of data size.

Operation (Op)	Line#	Cost/Op	#Op
1. Update mass l	8	th	n
2. Score($x, T.root$)	9	th	n
3. Change detection	14	$t\psi$	$\frac{n}{\psi}$
4. Update model	16	$t\psi$	$\frac{n}{\lambda\psi}$
5. Reset $N.l \leftarrow 0$	19	$t\psi$	$\frac{n}{\psi}$
Total cost, $\mathcal{T}(n) = O(2nt(h+1+\frac{1}{2\lambda}))$			

Table 2: Amortised Analysis for Streaming HS-Trees: Total time cost for n streaming points. Line# refers to the line number in Algorithm 5.

3.4 Change detection. The idea is to capture significant changes in mass within a subspace defined by branch in an HS-Tree. Given a subspace which has a high mass in a reference window. If this subspace has a significant change in mass in the latest window (compared to the reference window), then a significant change in normal points have effected. This is because a high-mass subspace is normally associated with normal points. Changes in the high-mass subspaces indicate changes in distribution due to the normal points (i.e., $P(x|y = normal)$). When this occurs, the model shall be updated to the latest mass profile. We give a more precise definition of our change detection method below.

Let L be a set of $Node$ with non-zero (reference and latest) mass of HS-Trees; that is:

$L = \{Node : (Node.r > 0) \vee (Node.l > 0)\}$

The mean of all non-zero mass $Node.r$ of HS-Trees is given as:

$$\overline{r} = \frac{1}{|L|} \cdot \sum_{Node \in L} Node.r$$

The set of high-mass $Node$ is defined as:

$$L_{high} = \{Node : (Node \in L) \wedge (Node.r > \overline{r})\}$$

L_{high} contains the mass profile of a set of subspaces with high mass. Examining the changes of this profile allows us to describe the changes which have taken place in a data distribution.

The percentage of change in the high-mass profile of a latest window with respect to that of a reference time window is given as follows:

$$d = \frac{\sum_{Node \in L_{high}} |Node.r - Node.l|}{\sum_{Node \in L_{high}} Node.r}.$$

3.4.1 Change detection for model update. We now define the required amount of change in the high-mass profile to be considered as 'large enough' to update a model. Let d_ω be the percentage of change in the high-mass profile, where ω is the index of each time window.

We estimate the average of d using an exponential update rule:

$\hat{d}_{\omega+1} = \alpha \cdot d_\omega + (1-\alpha) \cdot \hat{d}_\omega$

Here, α is the smoothing constant in the range of $[0, 1]$ and a higher α-value gives more weight to recent observations.

The average deviation δ from the average of d is defined similarly:

$\hat{\delta}_{\omega+1} = \alpha \cdot |\hat{d}_\omega - d_\omega| + (1-\alpha) \cdot \hat{\delta}_\omega$

Let $d_{\omega+m}$ be a change that occurs at m number of windows after its reference time window ω. This change is considered large if

$d_{\omega+m} > \hat{d}_\omega + \tau \cdot \hat{\delta}_\omega$,

where $m \geq 1$ and τ define the amount of deviation that is considered to be 'large enough'.

The model is only updated if changes are detected in λ consecutive windows. Based on this, a change is categorised as:

- a transient change if $m < \lambda$;
- a persistent change if $m \geq \lambda$.

3.4.2 Model updating schemes. The proposed model update scheme based on persistent change is called Selective Update scheme, denoted as SU. We will compare this scheme with two other schemes. The first scheme assumes that an initial model can be used throughout the entire stream. We denoted this scheme as NoU, which stands for No Update. The second scheme Always Updates (denoted as AU) the model at each time window.

Table 3 summarises the strengths and weaknesses of these three schemes under different conditions. NoU is expected to work well when there is no change in the distribution of normal points; otherwise NoU will fail because the initial model will become irrelevant after a change due to $P(x|y = normal)$ has occurred. AU is expected to cope well when there is a distribution change in the normal points; but it fails when the change is due to $P(x|y = anomaly)$ or $P(y = anomaly)$. SU is expected to work well in most cases, except when there is a transient change due to $P(x|y = normal)$. In this case, SU will miss to update its model and therefore will not perform well during the short period in which a transient change occurs.

Why a delay in λ windows is required before a model update? When there is a combination of changes involving both $P(x|y = normal)$ and $P(x|y = anomaly)$, a delay will avoid transient Type (I) change and still be able to detect anomalies over the duration of this change. Otherwise, the anomaly detector will fail to detect anomalies during this transient Type (I) change while adapting to it. Note that this failure can be very serious as the Type (II) change could

Type of Change	SU	NoU	AU
Type (I)	✓(persistent) ✗(transient)	✗	✓
Type (II)	✓	✓	✗
Type (III)	✓	✓	✗

Table 3: A summary of the working conditions for the three model updating schemes. ✓ means a scheme can cope with a change. ✗ means a scheme cannot cope with a change.

accompany a significantly increased number of anomalies (i.e., Type (III) change). We demonstrate this combination of changes using one example in Section 3.4.3 and its effect in Section 5.1.1.

3.4.3 Changes in high-mass profile when data distribution changes. The aim of this section is to provide a systematic analysis of four possible scenarios associated with distribution change in the context of anomaly detection. To facilitate our analysis, we will consider a synthetic dataset with two Gaussian clusters—a big normal cluster in which the centroid is at the origin; a small anomalous cluster with about 9% the size of the normal cluster, and its points are scattered. An example of the data distribution *before any changes occur* is as shown in Figure 1. This distribution will undergo a change in the middle of the streaming process.

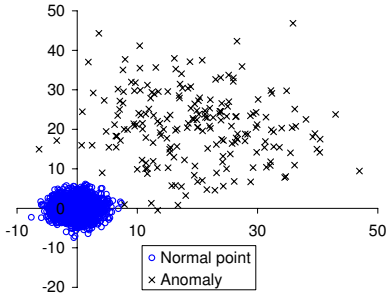


Figure 1: The original synthetic dataset consists of a big normal cluster and a small abnormal cluster.

The first case considered here is associated with distribution change due to normal points only (i.e., $P(x|y = normal)$, as shown in Figure 2. This case is concerned with a shift of the normal cluster to a completely new centroid location. An anomaly detection model must adapt to the new data distribution in order to maintain its detection accuracy.

The second and third cases are associated with changes in anomalies only. Figure 3 depicts these two cases. The second case is associated with a three-fold increase in the

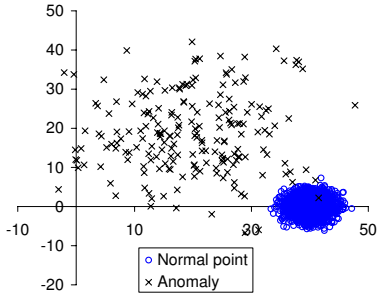
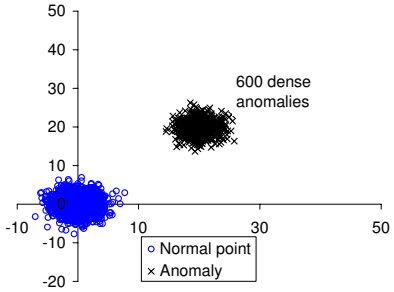
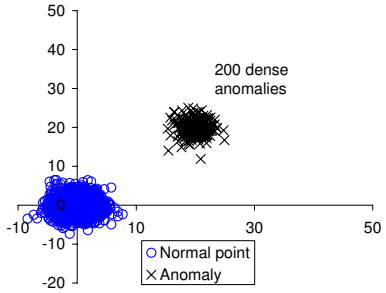


Figure 2: Case 1: A shift in normal cluster centroid. This is a Type (I) change due to $P(x|y = normal)$.

number of anomalies (i.e., $P(y = anomaly)$ as well as an increase in density (i.e., $P(x|y = anomaly)$. The third case involves an increase in density of anomalies only (i.e., $P(x|y = anomaly)$). These two cases should not trigger a model update because there is no change in the normal points.



Case 2: Anomalies increase in number and density



Case 3: Anomalies increase in density

Figure 3: Case 2 has Type (II) and Type (III) changes. Case 3 has Type (II) change only.

The fourth and the last case is as shown in Figure 4. This case is more complicated because it involves the following changes: (i) a shift of the centroid of the normal cluster, and (ii) an increase in the number and density of the anomalies, where the anomalies occur as short bursts during the streaming process. We expect a model update to occur

when there is a distribution change of the normal points. However, when there are short bursts of anomalies, we do not want to update the model.

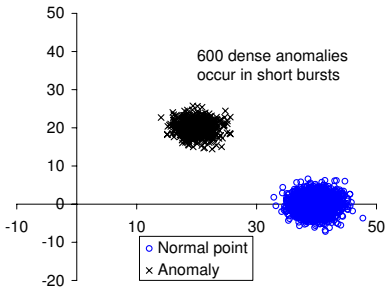


Figure 4: Case 4: A change of data distribution due to a combination of several changes in normal points and anomalies. These Type (I), Type (II) and Type (III) changes.

Figure 5 shows the changes in the high-mass profile associated with two synthetic cases identified earlier. It is easy to see that the high-mass profile changes quite significantly for Case 1 at the middle of the stream in which Type (I) change has occurred. When there are Type (II) and Type (III) changes which involve anomalies only, as in Case 2, the change in the high-mass profile is relatively small. These examples show that analysing the changes in the high-mass profile is an effective way to detect distribution change due to $P(x|y = normal)$, or Type (I) change.

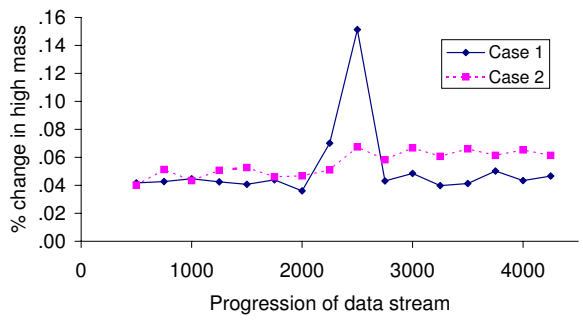


Figure 5: An example of high-mass profile changes in two different synthetic cases.

4 Experimental Setup

4.1 Data. We use the four synthetic datasets presented in Section 3.4.3. Each of these synthetic datasets has a very specific property and allows us to analyse the conditions under which a method could fail.

We also use six large datasets from different domains. The first two datasets are SMTP and HTTP from KDD CUP 99 network intrusion data as used in [13]. These datasets have a temporal aspect in their data sequence, which resembles the properties of streaming data. HTTP is characterised by its sudden bursts of anomalies in some streaming segments. SMTP does not produce bursts of anomalies, but possibly exhibits some distribution changes within the streaming sequence.

In practice, it is hard to quantify whether a distribution change has indeed occurred within a stream. For this reason, we have derived a dataset, SMTP+HTTP, which contains the SMTP data instances follow by the HTTP data instances. When viewed as an entire stream, we expect a distribution change to occur when the communication protocol is switched from SMTP to HTTP.

We also use the COVERTYPE and SHUTTLE datasets from the UCI Machine Learning Repository [2]. COVERTYPE is a relatively large dataset and is commonly used in data stream research. We split the anomaly class into several small groups and placed them in different segments of the dataset. As will be shown later, this simulates short bursts of anomalies in different streaming segments. As for the SHUTTLE dataset, it represents a situation where there is little or no distribution change.

The last dataset we use is MULCROSS [11]. This dataset contains dense clusters of anomalies that are harder to detect than scattered anomalies. We expect many traditional anomaly detection methods to fail in detecting the dense anomalies in this dataset.

Table 4 gives a summary of the real and synthetic data used in this study.

	N	D	anomaly class
Synthetic Case 1,3	4400	2	class 2 (9%)
Synthetic Case 2,4	4800	2	class 2 (16.7%)
SMTP (KDD Cup 99)	95156	3	attack (0.03%)
HTTP (KDD Cup 99)	567497	3	attack (0.4%)
SMTP + HTTP	662653	3	attack (0.35%)
COVERTYPE	286048	10	class 4 (0.9%) vs. class 2
SHUTTLE	49097	9	class 2,3,5-7 (7%)
MULCROSS	262144	4	2 clusters (10%)

Table 4: A summary of the characteristics of datasets used. N is the number of instances and D is the number of dimensions. The percentage in bracket indicates the percentage of anomalies.

4.2 Settings. For all the experiments, we conducted ten independent runs of each algorithms on each dataset. Each run was conducted as a single threaded job processed at 2.3GHz in a Linux cluster (www.vpac.org). Once the anomaly scores for all instances are obtained, the results can be evaluated by using the anomaly scores to rank the instances. Normal

points are expected to have high scores, whereas anomalies are expected to have low scores. From this ranking and the ground truth, we then compute the AUC (Area Under receiver operating characteristic Curve) [4] to measure the performance of all anomaly detectors reported in this paper. The actual CPU times are also reported for the six large datasets.

The parameter settings for Streaming HS-Trees are as follows. We use an ensemble size (t) of 25 trees. For the exponential update of changes in mass profile, we set the smoothing constant α at 0.3. The window size (ψ) is 250. The threshold (τ) for detecting a change in the high-mass profile is 4. The number of consecutive changes (λ) for persistent change is set at 1 for all the small synthetic cases. For the six large datasets, λ is fixed at 4. All these settings remain unchanged throughout the experiments.

5 Experimental Results

In this section, we report the results of two experiments. The aim of the first experiment is to examine the effectiveness of the change detector in Streaming HS-Trees for performing model updates in order to cope with evolving data streams. To demonstrate the practical benefits of the proposed method, we compare Streaming HS-Trees with the three model updating schemes described in Section 3.4.2, namely Selective Update (SU), No Update (NoU) and Always Update (AU). Results of this experiment will be reported in Section 5.1.

The aim of the second experiment is to examine how Streaming HS-Trees fares in comparison with existing anomaly detection methods. The benchmarking methods include ORCA - a distance-based anomaly method based on k-Nearest Neighbours (k-nn) [3] and One-Class Support Vector Machine (SVM) [12]. ORCA is selected because it is a major improvement of k-distance anomaly detection implementation in terms of efficiency. Using a pruning rule with randomly ordered samples, the time complexity of ORCA is reduced to near linear. The parameters of ORCA is ($k = 5$ and $N = n/8$) following the same treatment as in [10]. As for One-Class SVM, we apply the Radial Basis Function kernel and parameter setting as suggested in [5]. Results for this experiment will be reported in Section 5.2.

5.1 Results of Streaming HS-Trees with three model updating schemes. The presentation of results is organised as follows. Section 5.1.1 provides an analysis of the results on the four synthetic cases. Section 5.1.2 gives a detailed analysis of the results on large datasets. Section 5.1.3 gives an analysis of the speed of processing the largest dataset (i.e., SMTP+HTTP) used in this study.

5.1.1 Results on synthetic cases. Table 5 shows that Streaming HS-Trees using the SU scheme is the most robust in terms of the overall AUC scores. Its results are either ranked first or second for the synthetic data. Notice that the performance of Streaming HS-Trees using SU scheme is slightly lower than that of using AU scheme in Synthetic Case 1. This is because SU scheme updates its model at the next time window after a change was detected, this degrades its performance slightly. As will be demonstrated in Section 5.1.2, this delay in model update is quite small for large streaming data and therefore the effects on the detection accuracy is not substantial.

Table 5 shows that NoU only works well when there is no changes in the high-mass distribution (i.e., Cases 2 and 3). It fails badly when a major distribution change occurs in Cases 1 and 4. This is because the initial model used in NoU is no longer relevant after a distribution change in normal points has occurred in the middle of the stream.

AU works well when there is a distribution change in normal points only (i.e., Synthetic Case 1). However, its performance degrades when there is a sudden increase in the number of anomalies in Cases 2 and 3. This is because the model updates itself and treats some of the anomalies as normal points. Furthermore, AU's performance becomes poorer when the anomalies occur in short bursts during the streaming process, as in Case 4.

Dataset	SU	NoU	AU
Synthetic Case 1	**0.935**	0.519	**<u>0.989</u>**
Synthetic Case 2	**0.996**	**0.996**	0.940
Synthetic Case 3	**<u>0.997</u>**	**0.992**	0.990
Synthetic Case 4	**<u>0.982</u>**	0.531	**0.789**
HTTP	**<u>0.998</u>**	**0.984**	0.143
SMTP	**0.858**	0.753	**<u>0.874</u>**
SMTP + HTTP	**<u>0.994</u>**	**0.403**	0.262
COVERTYPE	**<u>0.915</u>**	**0.855**	0.743
SHUTTLE	**0.997**	**0.997**	**0.997**
MULCROSS	**0.974**	**<u>0.980</u>**	0.965

Table 5: A summary of the overall AUC scores for the synthetic and real data for Streaming HS-Trees using the three model updating schemes: NoU, AU and SU. The best result is boldfaced and underlined; the second best result is boldfaced.

5.1.2 Results on large streaming data. Table 5 shows that the performance of SU is the most consistent throughout all the datasets. We will discuss these results in conjunction with the AUC performance in different streaming segments in each dataset.

HTTP: AU performs a lot worse than both SU and NoU in this dataset. From our analysis summarised in Table 3, it is likely that there are TYPE (II) and/or TYPE (III) changes. Figure 6 reveals that the bursts of anomalies occur at segment 3 and the change continues to segment 4 to a lesser extend.

NoU and SU perform well in this dataset. NoU will never update its model and therefore detect the bursts of anomalies successfully. SU regards the bursts of anomalies as transient changes and never updates its model to these changes. Hence it also performs well on this dataset with the highest AUC of 0.998 as reported in Table 5.

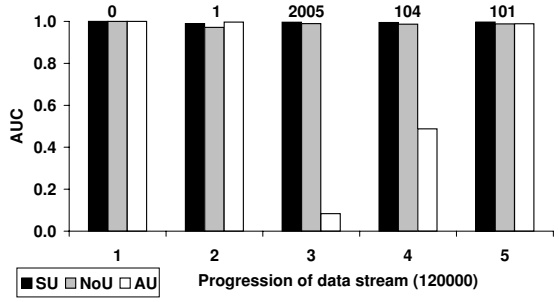


Figure 6: HTTP – AUC scores over five segments. The number on top of each segment is the total number of anomalies in that segment.

SMTP: AU and SU have similar performance in this data set, but NoU performs poorly. This scenario is likely to be due to Type (I) change. Indeed, Figure 7 shows that NoU has performed poorer than both AU and SU in all five segments. Segment 3 could be a result of a combination of changes in both transient Type (I) and Type (II) (or Type (III))—this causes both AU and SU to perform poorly as well.

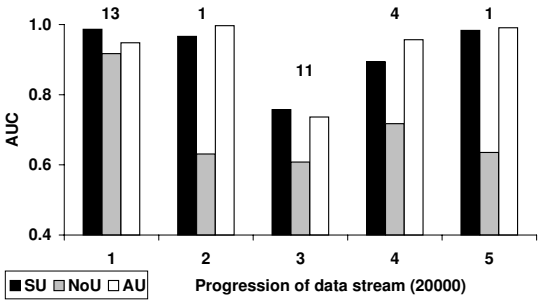


Figure 7: SMTP – AUC scores over five segments.

SMTP+HTTP: Because of a persistent change of $P(x|y)$ at the changeover from SMTP to HTTP in segment 1, NoU performs poorly throughout the HTTP stream from segment 3 to segment 5 in Figure 8. This is because NoU does not update its model that was previously learned from the SMTP stream in segment 1. The behaviour of AU and SU are consistent with their performance previously shown in Figures 6 and 7. Finally, SU has the best result in this dataset—its AUC remains high throughout the entire stream. This is because SU successfully updates its model when there is a change in the protocol, while avoiding to update its model when there is a burst of anomalies.

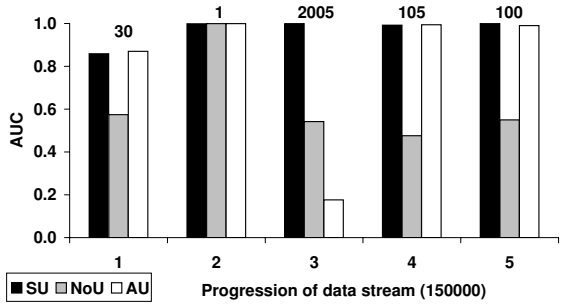


Figure 8: SMTP+HTTP – AUC scores over five segments.

COVERTYPE: This dataset demonstrates the scenario reminiscent to synthetic Case 4 in which there is a combination of persistent changes of Type (I), Type (III) and/or Type (II) in which only SU performs well out of the three schemes. Both AU and NoU notably perform poorer in segments 3 and 4 when there are bursts of anomalies as shown in Figure 9. SU suffers a little at the beginning of segment 1; this is likely to be due to one or two transient changes of Type (I).

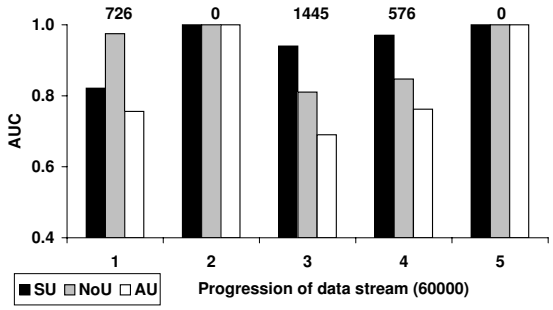
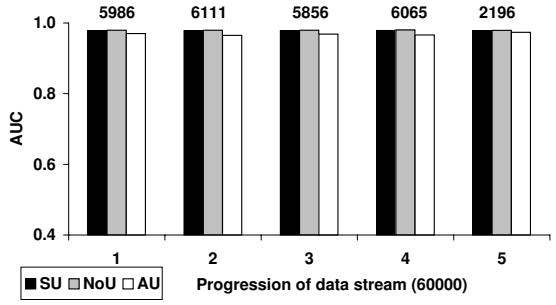


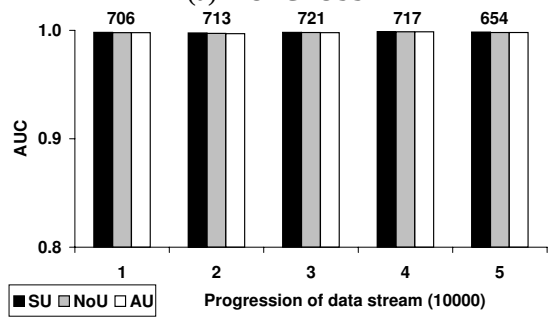
Figure 9: COVERTYPE – AUC scores over five segments.

MULCROSS and SHUTTLE: All schemes do well in these two datasets. It is likely that there is little or no changes in data distribution in them, as demonstrated in all segments of Figures 10 (a) and (b).

Our results show that SU can strike a balance between no adaptation to changes in data distribution, as in NoU; and over-adaptation, as in AU. As a result, Streaming HS-Trees adopting the SU scheme is more robust in evolving data streams. Table 6 shows that SU only performs a small number of model updates compared to the high number of updates performed by AU. For example, SU performs only 3 updates on the SMTP+HTTP dataset, compared to no update in NoU and 2650 updates in AU; yet its AUC performance is significantly higher than NoU and AU.



(a) MULCROSS



(b) SHUTTLE

Figure 10: MULCROSS and SHUTTLE – AUC scores over five segments.

	No. of Updates	
Dataset	**SU**	**AU**
HTTP	1	2269
SMTP	2	380
SMTP+HTTP	3	2650
COVERTYPE	1	1144
MULCROSS	1	1048
SHUTTLE	1	196

Table 6: Number of updates executed by SU and AU.

5.1.3 Analysis of stream processing speed. Figure 11 shows that, when processing the SMTP+HTTP dataset, Streaming HS-Trees can process at least 20,000 items per second. This is a reasonable speed for processing real streaming data or large static datasets. NoU is the most efficient scheme since it does not perform any model update. AU always update its model and thus it is not as fast as NoU. The speed of SU is in between the speeds of AU and NoU.

5.2 Comparison with ORCA and One-Class SVM. In this section, we compare Streaming HS-Trees (using the SU scheme) to ORCA and SVM. Table 7 shows that Streaming HS-Trees significantly outperforms ORCA and SVM, both in terms of AUC and runtime.

We see that Streaming HS-Trees attains an AUC score

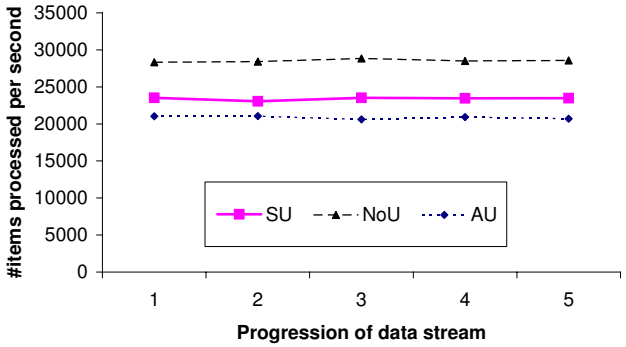


Figure 11: The number of instances processed per second over each segment in the data stream (SMTP+HTTP dataset).

of almost one (rounded to the nearest two decimal places) on the SHUTTLE, MULCROSS and HTTP datasets, whereas ORCA and SVM perform significantly poorer. In particular, ORCA does not perform well with MULCROSS and HTTP because of the density of the anomaly class is higher than normal instances. This reverses the ranking even to rank normal instances first before anomalies, causing a poorer than average AUC score.

Notice that One-Class SVM performs reasonably well on SMTP and HTTP, with AUC scores of 0.78 and 0.90 respectively. However, when these two datasets are combined as SMTP+HTTP, One-Class SVM fails quite badly. This is because the mixed distributions in SMTP+HTTP is too complex for One-Class SVM to handle, resulting in a poor AUC score of 0.43

Recall that Streaming HS-Trees is a one-pass algorithm, restricting itself to inspect each data point only once, whereas ORCA and SVM requires all the data to be load onto working memory. Table 7 shows that the efforts of building a one-pass Streaming HS-Trees translates well into a very fast processing speed. In contrast, One-Class SVM is the slowest because of the need to perform optimization during its model building process. ORCA is faster than SVM, but it is still (on average) 267 times slower than Streaming HS-Tree.

Another observation is that ORCA takes a longer time (13197 seconds) to process SMTP+HTTP, compared to the total time (267 + 9487 seconds) to process SMTP and HTTP datasets individually. This suggests that the processing time of ORCA becomes longer when the (combined) distribution becomes more complex. On the other hand, Streaming HS-Trees takes a relatively shorter time (35 seconds) to process SMTP+HTTP, compared to the total time (9 + 32 seconds) to process SMTP and HTTP datasets individually. This suggests that the processing time of Streaming HS-Trees is not sensitive to more complex data distributions.

	AUC			Runtime (seconds)		
	Streaming HS-Trees	ORCA	SVM	Streaming HS-Trees	ORCA	SVM
SHUTTLE	**1.00**	0.60	0.79	**7.09**	155.66	332.09
MULCROSS	**0.97**	0.33	0.59	**18.14**	2512.20	7342.54
HTTP	**1.00**	0.36	0.90	**32.07**	9487.47	35872.09
SMTP	**0.86**	0.80	0.78	**9.07**	267.45	986.84
SMTP+HTTP	**0.99**	0.38	0.43	**35.10**	13197.04	34918.70
COVERTYPE	**0.92**	0.83	0.90	**21.13**	6995.17	9737.81

Table 7: Streaming HS-Trees (using SU scheme) performs favourably to ORCA and SVM, both in terms of AUC and total processing time. Boldfaced entires are the best results.

6 Discussion

Many methods in data stream research employ the sliding window approach, for example, Concept-adapting Very Fast Decision Trees algorithm (CVFDT) [8]. Like Very Fast Decision Trees algorithm (VFDT) [6], CVFDT uses Hoeffding bound as a change detection mechanism to determine whether a model update is warranted. Here we highlight two key differences in comparison with Streaming Half-Space Trees. First, the change detection mechanism in CVFDT does not take into account the types of change and the two categories of change (i.e., transient change and persistent change) we have discussed in this paper. While one may argue that this is not required in the classification tasks, we suspect that this consideration will bring about more insights into data stream issues in classification tasks, especially in skewed class distribution problems.

Second, most sliding-window-based methods are known to be sensitive to the window size: if the window is too large, the model will perform poorly when there is a change in data distribution; if the window is too small, then the model will be inaccurate because of small training data size. This applies to CVFDT. As shown by our results, Streaming HS-Trees only needs a small amount of training data in order for an ensemble to perform well. The results of MULCROSS and SHUTTLE in Table 7 shows that Half-Space Trees performs significantly better than ORCA and SVM, even though ORCA and SVM employ all available data (50,000 and 260,000 instances) for training whereas each Half-Tree is trained from 250 instances only. Note that these two datasets have no change in data distribution which is the perfect condition for both ORCA and SVM.

There are some change detection methods proposed in the literature, e.g., the dynamic weighted majority algorithm (DWM) [9], and other variants [7]. DWM maintains an ensemble of base learners in classification tasks and predict using a weighted majority vote; and it dynamically creates and deletes base learners in response to changes in prediction performance. Thus, the change detection mechanism is solely based on the prediction accuracy to weight and discard each learner. Gao et al. [7] uses a data set with balanced distribution to train each model in the ensemble to deal with skew class distributions. Although these methods are generic change detection methods, it is unclear they can deal with window size issue; and because they do not assess the data distribution change directly, they are unable to deal with different types of change.

A framework for on-demand classification of evolving data streams [1] enables simultaneous training and testing streams to be used for classification. In contrast, our Streaming Half-Space Trees uses the same stream for training and prediction.

The key limitation of Streaming HS-Trees is space complexity which is exponential to the tree height h. The flip side of this limitation has many advantages, i.e., the model structure can be built without training data, and it only needs to be built once; the model structure does not change and the memory space stays constant throughout the entire data stream. An alternative is to build a model when the training data is available, and retrain another one if it needs to be updated. While this may reduce the overall memory requirement, it takes up previous time for training a new model in every model update; thus reduce the number of instances that can be processed—this can be critical in many real-world applications.

We have shown that the Selective Update scheme takes the best from two other schemes: No Update and Always Update. However, it is important to be aware of the constraint imposed by λ consecutive windows to define a persistent change: (i) the anomaly detector will perform poorly within the λ windows under type (I) persistent change. But this is limited to λ windows only—an error guarantee in the SU scheme that cannot be found in the other two schemes. (ii) If there are many transient type (I) changes, then SU will perform poorly. In practice, a data distribution change often involves a combination of different types of change which we have demonstrated that SU is more robust than AU and NoU in real-world scenarios.

7 Concluding Remarks

The proposed anomaly detection algorithm, Streaming HS-Trees, satisfies the key constraints for mining evolving data streams:

- It is a one-pass algorithm with amortised $O(1)$ time complexity and $O(1)$ space complexity—this allows it to deal with huge datasets or infinite data streams.
- It incorporates three mechanisms: anomaly detection, change detection, and model update, in a single framework.

The use of Half-Space Trees in the framework brings about the following features:

1) An HS-Tree structure can be built without any data.
2) Data profile can be updated incrementally in the HS-Tree structure as the data stream progresses.
3) The HS-Tree structure only needs to be constructed once and use throughout its entire life span, even when model updates are required during the streaming process.
4) Model updates are simple and efficient.

Most existing algorithms have only one or two of the above-mentioned features; incorporating all of the above features within a single method is a rarity.

We have identified the specific type of change in data distribution in evolving data streams that requires model update in order to maintain high detection accuracy. We have also identified other types of change that should not trigger a model update; otherwise the detection performance will degrade. This has led us to devise an effective change detection and model update mechanism in the framework.

We have shown in our empirical evaluation that Streaming HS-Trees with the selective update scheme is more robust in various scenarios in evolving data streams than two other schemes: always update and no update. We have also shown that Streaming HS-Trees significantly outperforms two state-of-the-art anomaly detection algorithms in terms of both detection accuracy and runtime.

References

[1] C. Aggarwal, J. Han, J. Wang, and P. S. Yu, *A Framework for On-Demand Classification of Evolving Data Streams*, IEEE Transactions on Knowledge and Data Engineering, 18(5) (2006), pp. 577–789.

[2] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[3] S. D. Bay and M. Schwabacher, *Mining distance-based anomalies in near linear time with randomization and a simple pruning rule*, Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press (2003), Washington, D.C., pp. 29–38.

[4] A. P. Bradley, *The use of the area under the ROC curve in the evaluation of machine learning algorithms*, Pattern Recognition (1997), 30 (7), pp. 1145-1159.

[5] B. Caputo and K. Sim and F. Furesjo and A. Smola, *Appearance-based object recognition using SVMs: which kernel should I use?*, Proc of NIPS workshop on Statitsical methods for computational experiments in visual processing and computer vision, Whistler (2002).

[6] P. Domingos and G. Hulten, *Mining high-speed data streams*, Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (SIGKDD00), ACM New York (2000), pp. 71-80.

[7] J. Gao, W. Fan, J. Han and P. S. Yu, *A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions*, Proceedings of 2007 SIAM International Conference on Data Mining (SDM'07), Minneapolis, MN, April 2007.

[8] G. Hulten, L. Spencer and P. Domingos, *Mining Time-changing Data Stream*, Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data mining, (2001) pp. 97-106.

[9] J. Z. Kolter, and M. A. Maloof, *DynamicWeighted Majority: An Ensemble Method for Drifting Concepts*, Journal of Machine Learning Research, 8 (2007), pp. 2755–2790.

[10] T. Liu, K. M. Ting and Z. H. Zhou, *Isolation Forests*, Proceedings of the 2008 IEEE International Conference on Data Mining, (2008), pp. 413–422.

[11] D. M. Rocke and D. L. Woodruff, *Identification of outliers in multivariate data*, Journal of the American Statistical Association, 91(435) (1996), pp. 1047-1061.

[12] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor and J. Platt, *Support Vector Method for Novelty Detection*, Advances in Neural Information Processing Systems, 12 (2002), pp. 582–588.

[13] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne, *Online unsupervised outlier detection using finite mixtures with discounting learning algorithms*, Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press (2000), pp. 320-324.